Neural Networks for Dynamic Flight Control

THESIS

Ronald E. Setzer
Captain, USAF

AFIT/GE/ENG/93D-36

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/GE/ENG/93D-36

Neural Networks for Dynamic Flight Control

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Ronald E. Setzer, M.B.A, B.S.E.E.

Captain, USAF

December 1993

Acknowledgements

Ronald E. Setzer

I would like to express my gratitude to my thesis advisor, Dr. Steven K. Rogers, for his guidance. I'd also like to thank the members of my thesis committee, Dr. Dennis Ruck and Dr. Mark Oxley. A special thank you goes to Dr. Matthew Kabrinsky for his encouragement and counsel.

To all the guys in the Comm Lab, thanks for making my stint at AFIT a little more enjoyable.

To Darrell Duane, my contact at George Mason University, thanks for all the E-mail that helped me over some major obstacles with FAME.

Finally, to my future wife Vicki, thank you for all your love, understanding and devotion during these past 18 months.

Table of Contents

Page

Page

List of Figures

AFIT/GE/ENG/93D-36

Abstract

This thesis examines the application of artificial neural networks (NNs) to the problem of dynamic flight control. The specific application is the control of a flying model helicopter. The control interface is provided through a hardware and software test bed called the Fast Adaptive Maneuvering Experiment (FAME). The NN design approach uses two NNs: one trained as an emulator of the plant and the other trained to control the emulator. The emulator neural network is designed to reproduce the flight dynamics of the experimental plant. The controller is then designed to produce the appropriate control inputs to drive the emulator to a desired final state.

Previous research in the area of NNs for controls has almost exclusively been applied to simulations. To develop a controller for a real plant, a neural network must be created which will accurately recreate the dynamics of the plant. This thesis demonstrates the ability of a neural network to emulate a real, dynamic, nonlinear plant.

Neural Networks for Dynamic Flight Control

I. Introduction

1.1 Overview

Automated aircraft control is obviously a vital concern to the US Air Force. Higher aircraft performance requires an increasing reliance on automated control systems. No where is this more evident than in helicopter control. Inherently unstable, helicopters are difficult to control under the best of circumstances. When performance is stretched to the limit, instability problems are often beyond the capability of the pilot.

With the ability to model non-linear, dynamic systems, artificial neural networks (NNs) are well suited for application to automated helicopter control. The problem now becomes one of designing, testing, and implementing a NN-based helicopter controller.

The heart of the problem lies in application of neural network-based control system to a real plant. Research done by others (23, 11, 19) was accomplished using computer simulations of plants. The difficulty in applying these approaches to the control of real plants is in the mathematical modeling necessary to develop computer simulations. One approach, devised by Nguyen and Widrow (13) holds more promise for application to a real plant. This self-learning system uses two NNs, one to control the plant and another, a plant emulator, which is used to train the controller NN. The emulator NN is trained to accurately reproduce the performance of a complex, nonlinear plant.

The Nguyen application also used a computer simulation of a plant. The actual plant dynamics were used in conjunction with the emulator to train the controller. In an application to a real plant, mathematical representation of the plant dynamics will not be available to train the controller NN.

In theory, a well trained emulator NN can successfully be used to train a controller NN. Thus, the key to using this method is to successfully train a neural network to emulate a real, nonlinear dynamic plant, such as a helicopter. Only after this research is accomplished can the task of developing a controller NN be attempted.

Full scale research is both time consuming and costly. Simulations require extensive mathematical modeling and computer resources. A more practical method is to scale down the system to permit experimentation in a laboratory environment. Through the use of a flying, scale-model helicopter, the development of a NN-based controller that can maintain a helicopter in a stable flight configuration becomes much more realizable. The first step in this process is to fully emulate the helicopter's performance parameters in a neural network.

1.2 Background

1.2.1 Neural Networks. An artificial neural network, sometimes simply referred to as a neural net (NN), is a system designed to emulate the functions, or supposed functions, of a biological neuron. While some may propose a NN is an artificial brain, the fact is there is no conclusive evidence that an artificial neural net is in any significant way a representation of a brain. However, NNs have been shown to possess unique characteristics such as the ability for function approximation and the potential for parallel implementation (1:18).

A simple example of a neural net is the perceptron (Figure 1). The perceptron inputs are weighted and then summed. With proper weights installed, the perceptron is capable of simple classification tasks.

The natural evolution of the perceptron is the multilayer neural network (Figure 2). The one property of multilayer neural networks which is central to most applications in control is that such networks can generate input/output maps which can approximate, under mild assumptions, any function with any desired accuracy (1:8).

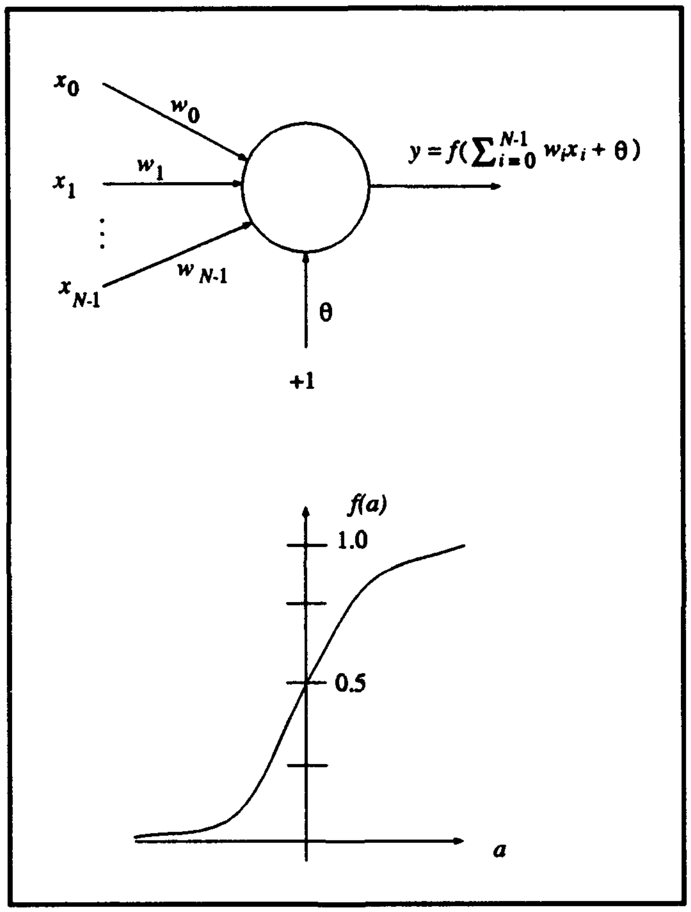


Figure 1. Perceptron (16:54)

The process of establishing the proper weights for a particular task is called training. The most common algorithm, or paradigm, used is called the backward propagation learning rule, sometimes just "backprop" (16:54). The method basically compares the output generated with an arbitrary set of weights to the set of desired outputs. The backpropagation algorithm corrects the weights based on the difference between the actual and the desired output.

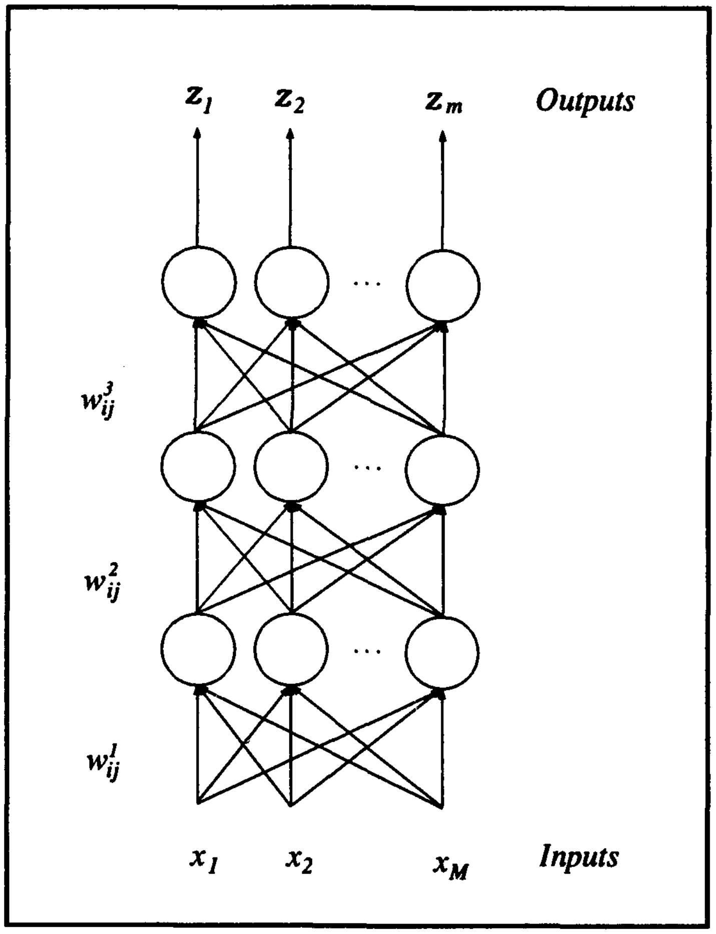


Figure 2. Multilayer Perceptron (16:54)

1.2.2 Helicopter Stability and Control. Helicopters are inherently unstable devices. The design of the helicopter gives it a great deal of maneuverability, but also makes it very difficult to

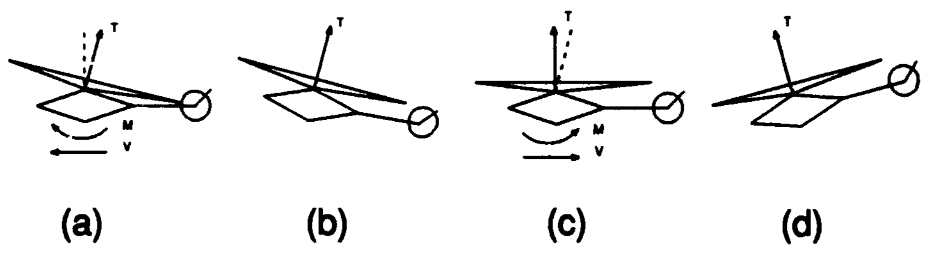


Figure 3. Suppose the hovering helicopter to experience a small forward velocity as at (a). Incremental flapping creates a nose-up disc tilt, which results in a nose-up pitching moment on the aircraft...A nose-up attitude develops and the backward-incline thus opposes the forward motion and eventually arrests it, as in (b)...A backward swing commences...exerting a nose-down moment, as at (c). A nose-down attitude develops and the backward movement is ultimately arrested, as at (d). The helicopter then accelerates forward under the influence of the forward inclination of thrust and return to the situation at (a). Mathematical analysis shows, and experience confirms, that the motion is dynamically unstable, the amplitude increasing steadily if the aircraft is left to itself. (20:127)

control. A classic remark made by a student following his first attempt to hover was, "It's like riding a pogo stick over a floor covered with greasy ball bearings" (14:542). The instability of helicopters is further illustrated in J. Seddon's description of Figure 3.

In certain situations, such as when carrying a suspended or slung load, the oscillations can become much more pronounced. This situation develops due to two factors. First, the helicopter's reaction to control is often delayed. In general, a helicopter without stability augmentation provisions is not only unstable, but its response to control input is slow (14:542). Second, a pilot often cannot physically react quickly enough to correct the problem. In human subjects carrying out a task on command, activity in the association cortex takes place about 200 to 300 milliseconds prior to movement (18:271). By the time a pilot reacts, he is often compounding rather than correcting the problem. This situation is commonly referred to as pilot induced oscillation.

With reaction times measured in microseconds, a computerized controller is theoretically capable of reacting much faster to these instability problems. However, conventional control theory limits the application of traditional computerized flight control systems to helicopters.

1.2.3 Automated Flight Control. Complex automated control systems have been a necessity in fixed-wing aircraft for years. For example, the F-16, intentionally designed to be unstable to obtain better maneuverability, is uncontrollable without computer assistance.

This problem is compounded in helicopters. The most notable distinction which emerges is that with fixed-wing aircraft, the stability equation leads to a simple physical interpretation of the motion, but with the helicopter this unfortunately is not so, and as a consequence it becomes a more complicated process. (20:126)

Traditional control theory has been inadequate for helicopter control except for limited applications. Autostabilizing systems have in the past used mechanical devices integral to the rotor (20:132). These systems are primarily designed to relieve the pilot of constant minute control corrections rather than act as true automated controllers.

The limitations in automated control are primarily due to the linear models used in conventional control theory. The control relationships are broken up into a series of linear approximations. As the system diverges further from the linear solution, the number of approximations increases. This produces two drawbacks: first, the problem becomes more difficult to correctly quantify, and second, the number of computations increases, slowing the response of the controller.

1.3 Resources

1.3.1 Fast Adaptive Maneuvering Experiment (FAME). The Air Force Office of Scientific Research, Mathematics and Computer Science branch, began the study of applying NN control to helicopter flight systems. The objective of this research is to automatically detect and correct flight instabilities such as those produced by an oscillating slung load. In the course of its efforts, AFOSR developed a small-scale test bed designed for NN experimentation called the Fast Adaptive Maneuvering Experiment (FAME).

The Fast Adaptive Maneuvering Experiment (FAME) is designed to provide neural network (NN) researchers with a physical, non-linear system of modest dimensionality with coupled dynamics (5:5). Developed by Dr. Kenneth Hintz at George Mason University under contract by AFOSR, FAME consists of a Kalt Whisper electric helicopter, an instrumented Flitemaster Jr. flight stand, and a Motorola MC68HC11 microcontroller unit (MCU) (Figure 4).

FAME also includes software to send commands to the control servos on the helicopter and to report helicopter attitude and position. Previous versions of the FAME software (3.0 - 5.0) relied on control inputs from the keyboard. This made manual control haphazard at best. Version 6.0 uses the control pad from a commercially available radio control (R/C) flight simulator called Skylark. This simulation package includes a control pad identical in configuration and operation to a standard R/C transmitter. This allows the researcher to manually fly the helicopter with greater precision. Precise control of the helicopter is necessary in order to gather accurate data needed for training the NN.

1.3.2 Neural Network Design. Several options for development of the neural network design are available. The two most promising are highlighted below.

Neural Graphics. Neural Graphics is a neural network simulator developed by Gregory Tarr during his Ph.D. dissertation (21). The initial plan was to use this software in

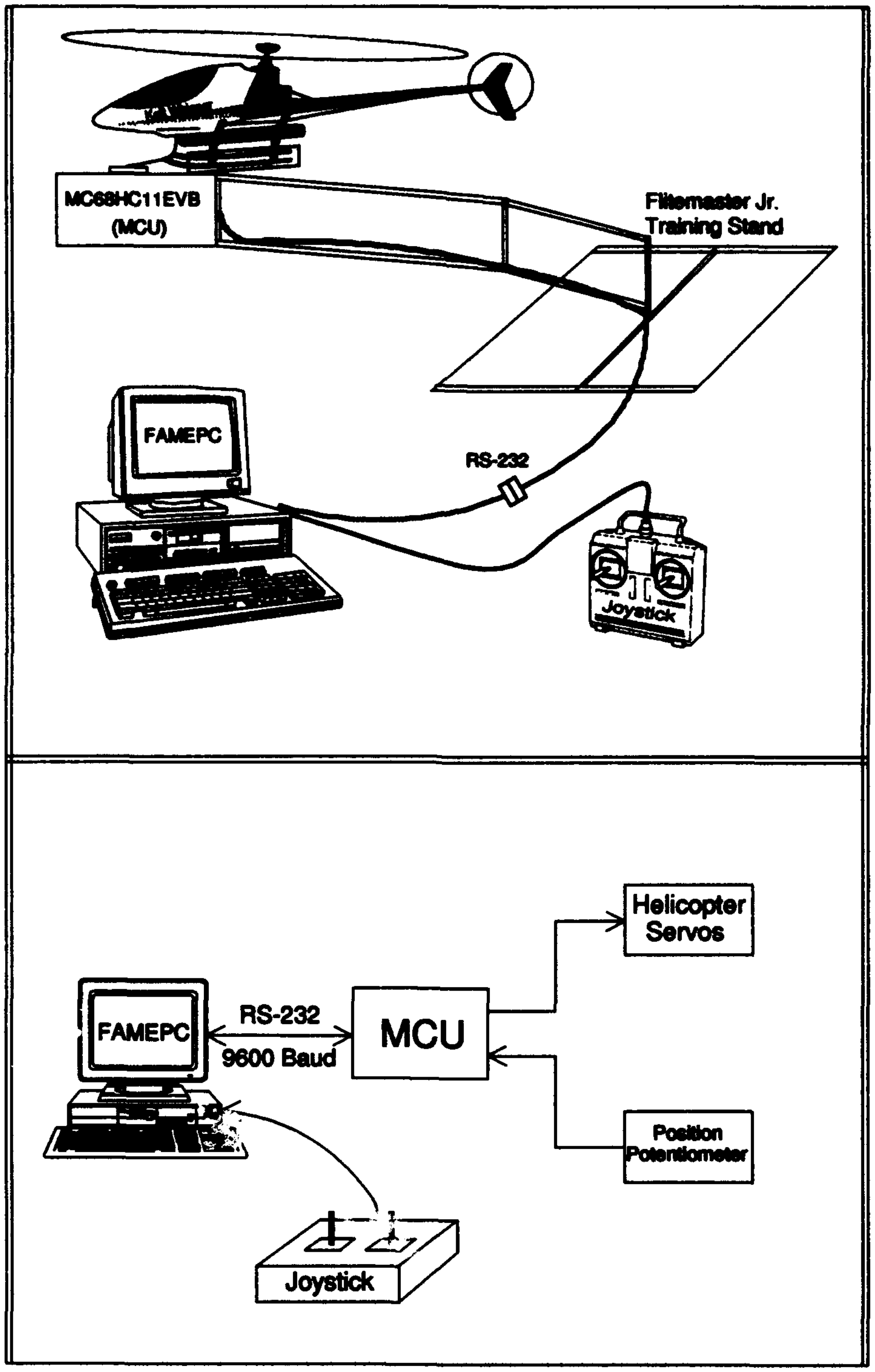


Figure 4. FAME System Components (5:6)

developing the NN helicopter controller. This approach is advantageous since there is still some corporate knowledge at AFIT about Neural Graphics. The primary disadvantage is that Neural Graphics was designed to be a simulator and primarily is a learning tool. However, there has been some work into practical applications of Neural Graphics.

Capt James B. Calvin, Jr. investigated classification of radar emitters using Intel's 80170NX chip, the Electronically Trainable Analog Neural Network (ETANN) in his master's thesis (2). In his work, he produced a modified version of Neural Graphics designed to simulate ETANN, a hardware-based neural network. He could produce the NN weights and run ETANN simulations by using Neural Graphics.

One approach is to follow Capt Calvin's work and use Neural Graphics to simulate the ETANN. By using this approach, it will be possible to solve the problem of producing an interactive system using Neural Graphics. In addition, the helicopter NN control design could be developed into an on-board NN controller by using the ETANN integrated chip. Although it is unlikely this research will directly produce an ETANN implementation, planning along these lines will make the eventual move to ETANN an easier task.

The ETANN has several advantages to software-based NNs. First, the ETANN is a much faster system. The analog structure permits information to literally flow through the net rather than be iteratively calculated by a computer. Second, since ETANN is an IC-based system, this would make it much more suitable to an eventual on-board NN application.

The primary drawback is ETANN weights are essentially fixed and cannot be easily updated. However, this should not pose a problem since the controller is not designed to be adaptive in real time.

The primary difference between this research and Capt Calvin's is the use of an IBM-PC based version of Neural Graphics, where Capt Calvin used the UNIX base system (Silicon Graphics). The biggest drawback is the lower computational power of the PC when compared to Silicon Graphics. The primary advantage is the PC is much less expensive and a more easily transportable system.

Mathematical Modeling Software. Another option is to use a mathematical modeling program, such as MATLAB®. This approach offers several advantages, including the direct manipulation of vector and matrices. Since it has a command level interpreter, the code does not have to be compiled prior to running. This allows changes to be quickly and easily implemented. The primary drawback is MATLAB® is rather slow when placed in iterative loops, such as those required in a NN implementation.

1.4 Assumptions

In any simulation, certain assumptions must be made. In this case, since the plant is a scale-model of an actual aircraft, the primary assumption is the model will adequately represent a full scale aircraft.

In particular, the effect of the instrumentation stand is considered to be minimal. While this is probably not a valid assumption, anecdotal reports from AFOSR indicate the stand only dampens the control response, making it perform somewhat more like a larger model helicopter.

Also, the assumption must be made that the helicopter can be controlled by a NN and that the NN control will be faster than human control. As stated earlier, part of the controllability issue is directly related to the human reaction time. The system must be capable of reacting faster if it is to be useful.

This is a reasonable assumption when considering ETANN typically has propagation time of 6 microseconds (2:17). While the NN simulation on Neural Graphics or MATLAB® will be directly related to processor speed and network size, it should be sufficient to exceed normal human reaction time. Using an INTEL 386 based PC with a clock speed of 25 Mhz, it should be possible to obtain a propagation time of less than 1 microsecond per node of the neural network.

1.5 Scope

The scope of this effort will be limited to the application of NN to dynamic flight control using standard classical backpropagation paradigms (16). It will not investigate different methods of backpropagation. Since the focus is on application to the control problem, it is doubtful whether another approach is necessary. The choice of training method is primarily based on the speed of training. The task here is to determine the feasibility of application of NNs to a control task. Whether using standard backprop or some other method would be irrelavent to this research.

1.6 Approach

The first step in approaching this problem is to use the FAME apparatus to gather the necessary training data for the NN. Manual control of the helicopter using the Skylark Simulator control pad will generate the training data.

At some point, the choice of the best method of interfacing to Neural Graphics or MATLAB® must be made. There are several possibilities. The first is to integrate the FAME software and neural network software into a single, interactive program. This is a possibility since Neural Graphics is C-based and MATLAB® uses MEX-files to allow commands to be called in a C-based program. The second choice is to use two computers, with NN software running independently of FAME. However, this would seem an unreasonable demand on already limited resources. The last and most reasonable alternative is to record training data using FAME, and then load the data into the backpropagation routine to train the NN. Once trained, the NN weights would be loaded into a forward propagation routine written into the FAME program.

The type and quantity of data required must be identified. The FAME test stand reports pitch, roll, yaw and X-Y-Z coordinate position. This data will obviously be used in NN training. However, necessity of the time derivatives of position inputs (velocity and acceleration, both linear and angular) must also be considered. Another possibility is to use delayed input and let the NN determine the time derived inputs.

As for the quantity of data, one rule states that using a minimum of three times the number of input features will give a rough estimate of how many training vectors are needed per class (Foley's rule) (16). This rule has been useful in pattern recognition problems using NNs, although it is still not clear on the applicability to this particular control problem. In this particular research, the quantity of data required will be determined experimentally.

1.7 Summary

A neural network is definitely a candidate for a flight control system. Its ability to model nonlinear, dynamic systems is well documented. While others have approached the problem of NN application to flight controls, they have only conducted limited simulations. None have applied their research even on a small scale to an actual flying platform.

The most promising approach to developing a NN-based controller for a real aircraft is a self-learning system. In this two-stage process, the key lies in the development of NN which accurately reproduces its flight dynamics.

The NN research at AFIT has focused primarily on pattern recognition problems, which are not, in general, directly applicable to control problems. On the other hand, the AFIT controls research has not approached the possibility of using neural networks. Research into this helicopter control problem can bridge the gap between these two areas, opening the door to new perspectives and unique solutions.

II. Literature Review

This chapter examines some of the current literature in the area of neural network applications to control problems. It begins by first reviewing the area of neural networks in controls in general, then examines the application of NNs to the specific area of flight control.

The particular application of concern is NN control of a physically realizable system, in this case, a flying scale model helicopter. This system is a particular challenge to automated control due to the extent of nonlinear relations between control input and aircraft response. The ability of NN to handle not only the nonlinearities of control, but to also "learn" the correct control maneuvers has made it a prime candidate for a flight control system.

Other researchers have examined the issue of NN control of aircraft. Narendra and Mukhopadhyay developed a NN control system using the dynamics of a helicopter as the plant (11). In his research, Schley used a simplified mathematical model of an aircraft landing in the presence of severe wind gusts (19). Unfortunately, both applications have only examined computerized simulations of aircraft.

2.1 Analysis and Application of Neural Networks for Self-Learning Control Systems

This section examines the self-learning neural network control system devised by Nguyen and Widrow (13). In this system, two separate neural networks (NNs), a plant emulator and a controller, are used. The plant emulator is first trained to predict the next state, or position, of the plant based on control and present position inputs. During the controller training process, the controller NN drives the trained emulator NN through a series of trials. In this process, the emulator's final position error is backpropagated through the emulator NN to produce an error to be used for the controller training.

The Nguyen system lends itself well to the problem of helicopter control. Since the emulator fully parameterizes the plant, the necessity of precise control data is negated. That is, when using a single NN, the controller network be trained in a single stage, requiring precise data on the control inputs that would drive the plant to the desired final state. This necessitates precise manual flight control, a rigorous and time consuming process requiring a highly skilled pilot.

This is similar to the controller developed for a turbogenerator by Wu and others (23). The primary difference is the Wu emulator and controller are trained simultaneously in the case of the turbogenerator. This control system uses two subnetworks, one for input-output (IO) mapping and the other for control. The IO mapper, or neural network mapper (NNM), uses the errors between the plant and the network output to update the training weights. Next the neural network controller (NNC) uses the updated NNM weights to modify its own weights. However, this application is again applied to a simulated plant.

2.2 Background: Truck Backer Upper

The controller in the Nguyen application backs a simulated tractor-trailer from some arbitrary point to the final desired location and orientation at a loading dock. The objective is to train the controller to produce the correct signal u_k to drive the plant to the desired final state z_d given the current state of the plant z_k (Figure 5).

The controller in this system is developed in a two-stage process. The first stage is a fairly straight forward backpropagation problem. The emulator NN is trained to map the state and control inputs to the correct next state of the plant. During training, the emulator NN is presented with a series of uniformly distributed random inputs and corresponding outputs of the plant (Figure 6). Using the error between the output of the network and the actual state of the plant, the emulator's network weights are updated using a standard backpropagation algorithm.

Stage two of the process is training the controller NN (Figure 7). First, the controller randomly drives the emulator through a series of K states. For each input from the controller, the emulator produces the appropriate next state of the plant. Eventually, the emulator arrives at a

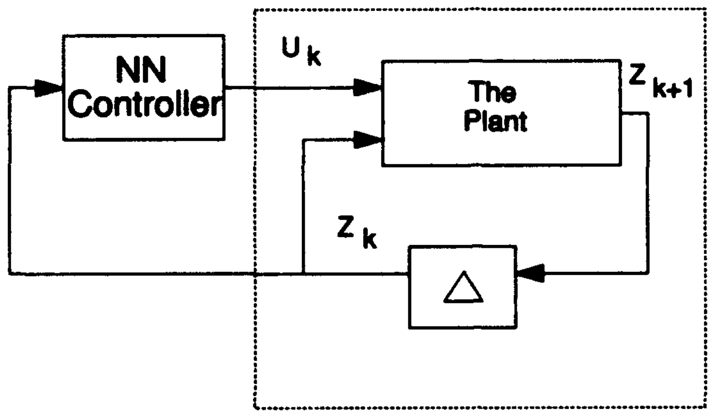


Figure 5. Plant and Controller (13:19)

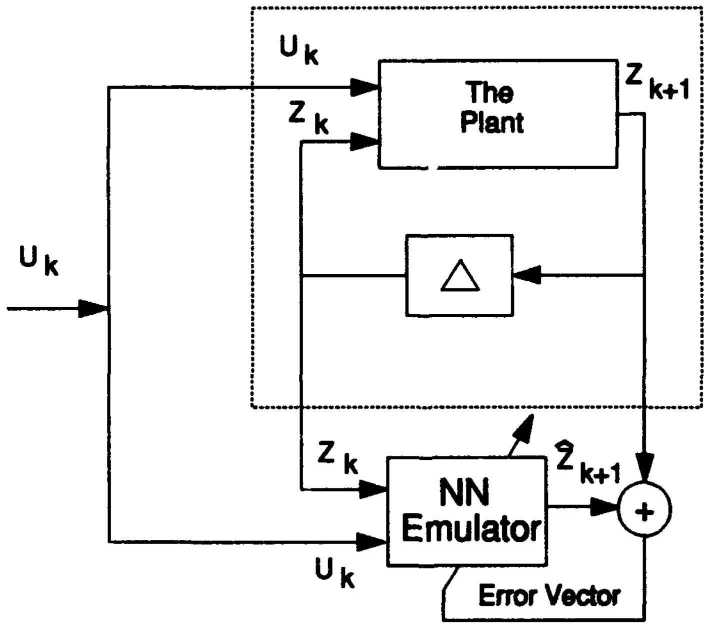


Figure 6. Training the Neural Net Plant Emulator (13:19)

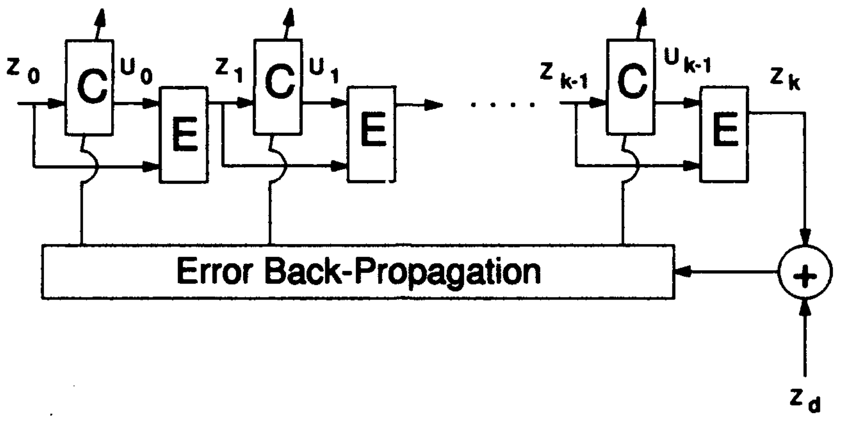


Figure 7. Training the Controller with Backpropagation (13:20)

predetermined stopping point (the Kth state). It is at this point where the emulator's final state is compared to the desired final state of the plant (position of the truck). Since there is no way to directly compare the controller's output to the final state of the plant, the state error must back-propagate through the emulator network in order to obtain the equivalent error at the controller NN output. The equivalent error can then be backpropagated through the controller to determine the appropriate weight updates.

During this process, it can be seen that the backpropagation through the emulator produces two error vectors: the control error and the state error. That is, the backpropagation reveals what control input should have been given and also what state, or position, was required to produce no error. This can be clearly seen when assuming the control input and state input to the emulator has no error. Then the output of the emulator would have no error and thus must be equal to the desired final state. In essence, the error in the prior state tells what state the emulator should have been in and the controller error tells the controller what it should have done.

2.3 Analysis

Up to this point, the process is fairly clear. Unfortunately, Nguyen purposely chose not to detail the error backpropagation through the emulator. This is unfortunate in that the backpropagation algorithm in the article does not accurately describe the process. More commonly referred to as back propagation through time (BPTT) (4, 8, 9, 15), it is not nearly as trivial a process as Nguyen implies. In particular, Nguyen does not detail the necessity of recording the output of the controller net at each time step during the training process. Without this information, the backpropagation algorithm will not work.

A report by Michael Lehr contains a more detained explanation of BPTT (8). This report clearly indicates the necessity storing the control output and hidden node outputs. Not only are the output weights required, but the output of the hidden layers must be available. The article mentions that the storage requirement can be lowered at the sacrifice of computations using the forward propagation algorithm to obtain the output of the hidden layer.

If we examine the backpropagation algorithm, we find as the error is backpropagated through each layer, the effect of the error decreases rapidly. Since BPTT essentially creates a neural net with nk hidden layers, the impact of the error quickly becomes insubstantial[1]. The problem manifests itself as a degradation of the error signal. As it backpropagates through layers of units its magnitude decreases; thus, the units far from the output receive a small degraded signal and take correspondingly longer to learn than those closer to the output (15:18). The same effect occurs when backpropagating through the emulator to obtain the control equivalent error. The amount of change to the controller weights will become almost infinitesimal.

Nguyen mentions the training method used involves starting close to the dock and then gradually moving further away. Although he doesn't detail why he chose this method, it does appear to be necessary when considering the backpropagation algorithm and the error propagation.

[1]n is the number of layers in the neural net and k is the number of time steps.

By starting only one or two states away from the final state, the final error has a much greater impact on the overall weight changes. As the initial state is moved further away, the weights in use are much closer to the final solution than those that would have been produced randomly. This appears to be a necessity when using BPTT. If he had started training at a greater distance from the dock, the backpropagation would have only had effect in the final two states. The error for states prior to this would have been near zero. The effect would have been to greatly increase the training time.

Subsequent correspondence with Derrick Nguyen (12) also revealed the details of backpropagation of the position error. When backpropagating, an equivalent error for the position vector is obtained from both the emulator and controller NNs. The question was which error is used for the prior state. The answer is both. The sum of both equivalent errors is fed to the previous state. Since the output of the previous state is connected to the input of both the emulator and the controller, the error at the output of a node is the weighted sum of the errors of all the nodes to which it is connected in the next layer. In this case, this would be both the position vector in the emulator and the controller. As it turns out, an analysis using the standard backpropagation algorithms would lead to the same result. The key is recognizing an output node of the previous state's emulator has an unweighted connection to one node in each of the next state's controller and emulator input layers (Figure 8).

For each training run, the weight changes for each state are calculated and then summed to be added to the controller's weights. However, in Nguyen's implementation, the controller's weights were updated as the changes are calculated. This would seem to cause problems since as the error backpropagates through each stage, the original weights would be constantly changing. But Nguyen points out that the changes for one run are so small as to not adversely effect the process. It is the accumulated effect over a large number of runs that improves the controller's performance.

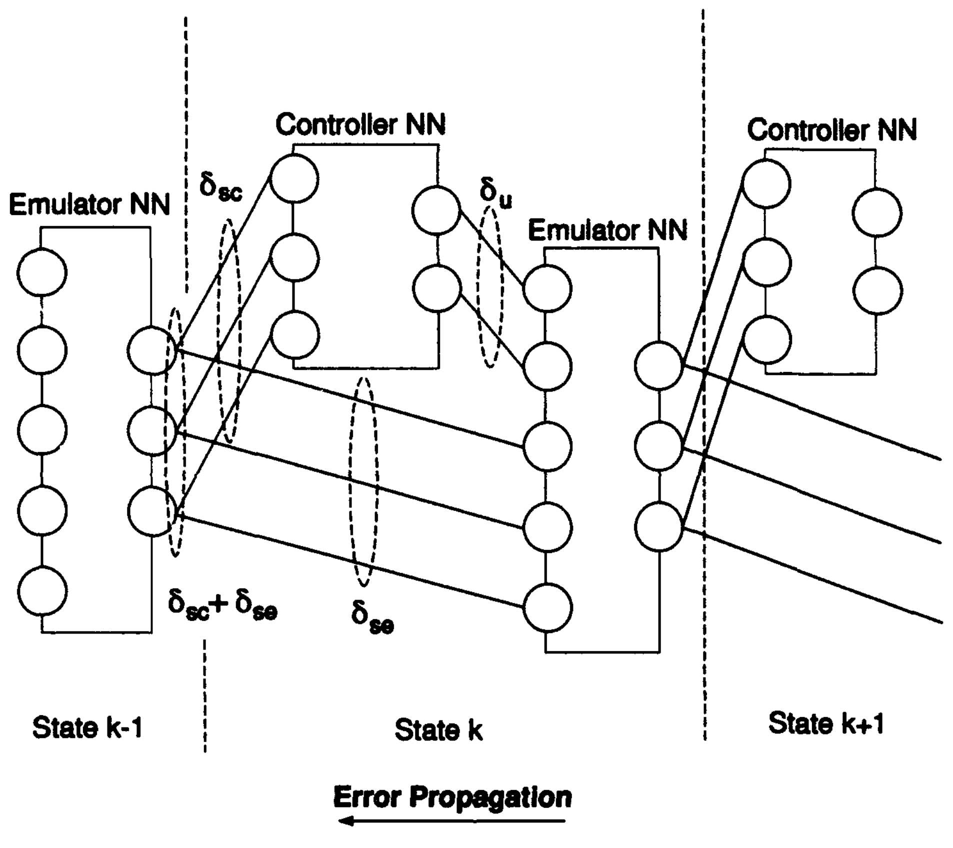


Figure 8. Equivalent Error Propagation. As the final position error backpropagates through time, the error presented to the controller (δ_u) in state k is the equivalent error at the control input to the emulator in state k. The error at the emulator in the previous state (k-1) is the sum of the position equivalent errors from the controller (δ_{sc}) and from the emulator (δ_{se}) in state k.

Care should be exercised with this procedure. It is possible for the network to lock in a partial solution. The net could be moving to a position in weight space where the error signals become so small as to make further movement impractical. This may be a feature of updating the weights after every example (15:18).

2.4 Backpropagation

Nguyen's application uses the Adaline (22) model for development of the neural networks. Neural Graphics and other AFIT applications use a slightly different model. The primary difference in the Nguyen application is that the Adaline uses the nonlinear function $f(\alpha) = tanh(\alpha)$ where standard backprop uses the sigmoid function $f(\alpha) = 1/(1 + e^{-\alpha})$.

The use of a sigmoid function should have little impact on the performance of the net, although some researchers attest that the hyperbolic tangent function might converge more quickly since the output range is -1 to +1, where the sigmoid ranges from 0 to +1.

2.5 Summary

The need to meet demanding control requirements in increasingly complex dynamical control systems under significant uncertainties makes neural networks very attractive. Their ability to learn, to approximate functions, to classify patterns, and their potential for massively parallel hardware implementation are the key characteristics. (1:8)

The Nguyen approach to developing a NN-based controller is the most appropriate for a flight control system. The emulator NN will fully parameterize the helicopter, negating the need for precise control data.

This method also presents implications to the development of simulator systems. The emulator completely parameterizes the plant without using mathematical derivations. A simulation of the device is possible since a complete I/O mapping is produced. Control inputs to an emulator NN will produce an accurate representation of the plant response.

While the training process may at first appear to be complex and time consuming, the final controller will consist of only a single multilayer NN. Lending itself to parallel hardware implementation, such as ETANN, the NN-based controller can react much faster than a conventional control system.

III. Methodology

3.1 Introduction

As mentioned in Chapter II, the Nguyen emulator/controller approach to developing a NN-based controller will be used in this research. However, prior to any attempt at developing the controller, an accurate emulator NN must be created. The development of the helicopter emulator will be approached in a two-stage process. The initial development of an emulator will focus only on the tail rotor of the helicopter. Using this simpler problem, the neural network design and training process can be refined prior to attempting the more complicated task of emulating the entire helicopter in the second step.

3.2 Stage One - Tail Rotor

The development of the emulator for the tail rotor is approached in a similar fashion to Nguyen's truck backer-upper. Since the FAME apparatus is capable of recording both input control and output position information, it is possible to record changes in the yaw position as the tail rotor control inputs are manipulated. Using this data, the emulator can be trained.

Initially, this approach included the use of PC Neural Graphics as the NN design and implementation software. It was later determined MATLAB® would be a more suitable design and experimentation tool. MATLAB® is especially efficient at processing vector and matrix information. The backpropagation equations (16) were easily converted into vector and matrix format (see Appendix C).

3.2.1 Emulator Development. The first attempt at development of a NN control system focuses only on the tail rotor control surface. Examination of a single control surface reduces the number of input and output parameters, thereby reducing the size and complexity of the network.

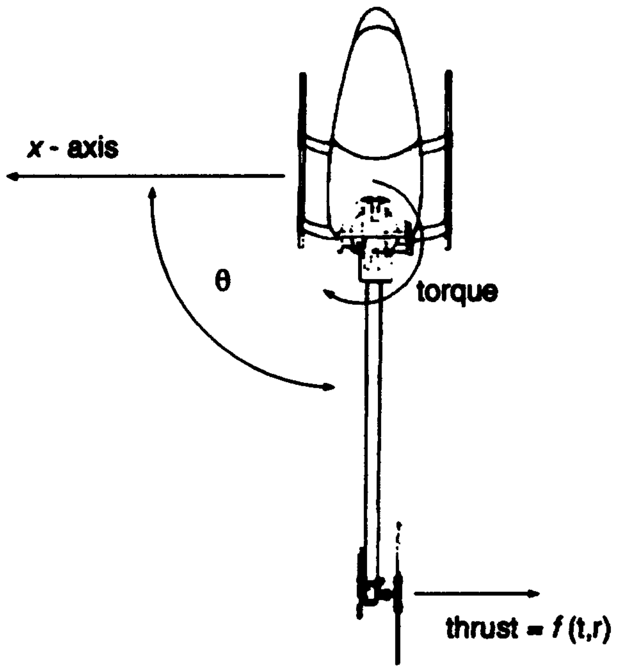


Figure 9. Tail Rotor Position. The tail rotor position (θ) depends on the force, or thrust, exerted by the tail rotor. The thrust is a function of the tail rotor blade pitch, or rudder (r) and the throttle setting (t).

The emulator NN is designed to map the input functions of state and control vectors to the output state vector. In this case, it was obvious the control vector would consist of at least the throttle setting and tail rotor pitch, represented by the variables t and r. These two inputs control the movement of the tail empanage (Figure 9).

The first step is to record data from the FAME apparatus. In order to isolate the tail rotor and reduce effects of motor torque, the main rotor is disable. This is accomplished by removing the gear from the main rotor shaft. Also, the training stand is adjusted so that movement about the pitch and roll axes is eliminated. Only yaw movement is permitted during this phase. Then, using the FAME joystick control, the helicopter is maneuvered manually through the entire range of throttle and rudder controls while measuring the yaw response.

The recorded data is normalized prior to training the network. The input and output data for training the emulator is normalized between 0 and 1. Since the FAME apparatus reports the various position and control information using an 8-bit binary number, the data is normalized by simply dividing the recorded flight data by 255 $(2^8 - 1)$.

It is necessary to determine the type of input and output data used in developing the controller. Along with the control inputs, the FAME system records the tail position or yaw angle with respect to time. With this timing information, it is also possible to derive the angular velocity and acceleration of the tail section. Although off-line calculation of these derivatives is possible, time delayed input samples should also provide similar results.

The initial set of input vectors selected included the throttle, tail rotor pitch, present yaw position, and the prior two yaw positions. This is represented in vector format as $[t,\ r,\ \theta_k,\ \theta_{k-1},\ \theta_{k-2}]^T$. Other input vector sets include the use of velocity and acceleration and also four and five position delay. These configurations will be referred to as the three-state, v-a, four-state and five-state models respectively.

3.2.2 NN Prototype. After recording, the data is transferred to a UNIX-based system (Sparc Station) and the emulator NN is prototyped using MATLAB®. Since MATLAB® is designed to process vector and matrix data, the backpropagation algorithms have been modified. The vectors $\mathbf{x}$ (m x 1), $\mathbf{a}$ (n x 1), $\mathbf{z}$ (p x 1),and $\mathbf{d}$ (p x 1) are the input, hidden layer output, actual output, and desired output vectors, respectively. The matrices $\mathbf{W}_i$ (n x (m+1)) and $\mathbf{W}_o$ (p x (n+1)) are the output and input weight matrices. The additional column in each weight matrices is due to the bias term which is appended to the input and hidden layer vectors.

Several different NN configurations are tested. The basic configuration consists of a single hidden layer using a sigmoid function and a linear input and output layer (linear-sigmoid-linear) (Figure 10).

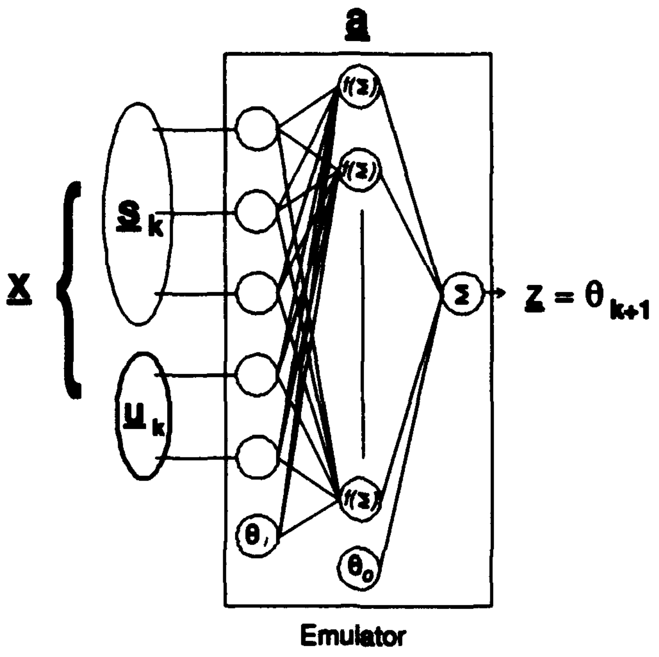


Figure 10. Linear-Sigmoid-Linear NN

The forward and backpropagation equations used are given below. A complete derivation can be found in Appendix C. The input vector is represented by **x**, the output vector by **z**, the hidden layer output by **a** and the desired output **d**. The equations for forward propagation through the network are

$$\mathbf{z} = \mathbf{W}_o\tilde{\mathbf{a}} \tag{1}$$

$$\text{where } \tilde{\mathbf{a}} = \begin{bmatrix} \mathbf{a} \\ \hline 1 \end{bmatrix} \tag{2}$$

$$\text{and } \mathbf{a} = f_h(\mathbf{W}_i\tilde{\mathbf{x}}) \tag{3}$$

$$\text{and } \tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \hline 1 \end{bmatrix} \tag{4}$$

In this case, the vector valued function $f_{\mathbf{h}}(\underline{\alpha})$ represents an elementwise operation on the n x 1 vector α defined as

$$f_{\mathbf{h}}(\underline{\alpha}) = \left[(1+e^{-(\alpha_1)})^{-1},\ (1+e^{-(\alpha_2)})^{-1},\ \ldots,\ (1+e^{-(\alpha_n)})^{-1}\right]^T \tag{5}$$

The weight update equations used for backpropagation are

$$\mathbf{W}_o^+ = \mathbf{W}_o^- - \eta \frac{\partial E}{\partial \mathbf{W}_o^-} \tag{6}$$

$$\text{where } \frac{\partial E}{\partial \mathbf{W}_o^-} = -(\mathbf{d} - \mathbf{z})\tilde{\mathbf{a}}^T \tag{7}$$

$$\mathbf{W}_i^+ = \mathbf{W}_i^- - \eta \frac{\partial E}{\partial \mathbf{W}_i^-} \tag{8}$$

$$\text{where } \frac{\partial E}{\partial \mathbf{W}_i^-} = -[diag(\mathbf{a} \odot (\mathbf{1} - \mathbf{a}))|\underline{0}_n]\mathbf{W}_o^{-T}(\mathbf{d} - \mathbf{z})\tilde{\mathbf{x}}^T \tag{9}$$

and $\odot$ is the Hadamard, or array, product (10)

The training process will iterate through the forward propagation and weight update equations, comparing the output of the network for each input control and state vector to the actual next state of the system. The error is used in the backpropagation equations to update the network weights.

3.2.2.1 Testing. To test the emulator NN, exemplars of the tail rotor state, or position, that have not been presented during training are fed through the forward propagation network. The output of the NN is then compared to the actual response of the tail rotor to the control inputs.

3.2.3 Controller Development. Continuing with the Nguyen approach, the output of the controller net is fed to the emulator along with the state information. The output of the emulator net is then compared to the desired final state of the plant. The error is backpropagated through the emulator net and produces an equivalent error at the input to the emulator net, which is also the output of the controller net. This equivalent error is then backpropagated through the controller net and the controller weights are updated.

In this case, the problem should be simpler than the truck backer upper. Unlike the truck, at each state in the tail rotor problem the desired state is always the same. At each state, the controller will always attempt to minimize the angular error. In the truck backer upper, back propagation through time (BPTT) is necessary since there is no direct path from a particular state to the final desired state. In order to minimize the final error, the truck might initially have to move in a direction away from the dock. In the tail rotor, each state has a direct path to the final state. At any instance, the controller would seek to minimize the angular offset from the final desired position. This removes the need for BPTT.

Nguyen also used the actual plant dynamics in the controller training process. This was possible since the plant used was a simulation, not an actual tractor-trailer. At each time step during forward propagation, the output of the controller was fed to both the emulator and the truck simulator, producing both and estimated and an actual next state. The actual next state is then presented to the controller. The estimate is not used until the final state. Thus, at each time step, the controller has an accurate representation of the plant's state. In this research, and in the real world, the plant dynamic will not always be available in mathematical form. The question remains whether this process will work for a real plant, without using a mathematical representation of its dynamics.

During the controller training process, a random present position vector is input to the controller. This in turn produces some random control vector. The control vector along with the present position vector is input to the emulator. The emulator NN then produces the next position of the plant. The error between this next state and the desired final state of the plant is backpropagated through the emulator to obtain the equivalent control and position errors at the input layer. The control equivalent error is backpropagated through the controller NN to update the network weights. In BPTT, the position equivalent error at the emulator input and the position equivalent error at the controller input would be summed to obtain the equivalent error for the previous state.

This process will continue until the error is reduced to an acceptable level. This level must be determined experimentally since, in most cases, the angular error cannot be reduced to zero in a single time step. However, the marginal error, or changes between sequential error measurements, should approach zero as the controller network becomes fully trained.

Testing the controller consists of presenting a random initial position to the controller NN and then observing the output of the emulator NN. If adequately trained, the controller NN should drive the emulator NN to the desired final position.

3.3 Stage Two - Helicopter Emulator

Development of the emulator for the remaining control and state inputs will be essentially the same as the tail rotor development. Again, the flight data recorded from the FAME apparatus is used to train the helicopter emulator. Exemplars now include all position information (pitch (ϕ), roll (ρ), yaw (θ), x-coordinate, y-coordinate, altitude (z)) (Figure 11) and the all control inputs (throttle, rudder, aileron, collective, cyclic) (Figure 12).

3.4 Controller Implementation on Actual Helicopter

Once the NN controller has been fully developed and tested, the next step is controlling the actual plant. FAME makes this fairly straightforward. Minor changes to the FAMEPC C-code (Appendix A) used in FAME should allow incorporation of the NN controller. A forward propagation routine would replace the manual control function "SetServoFromStick()."

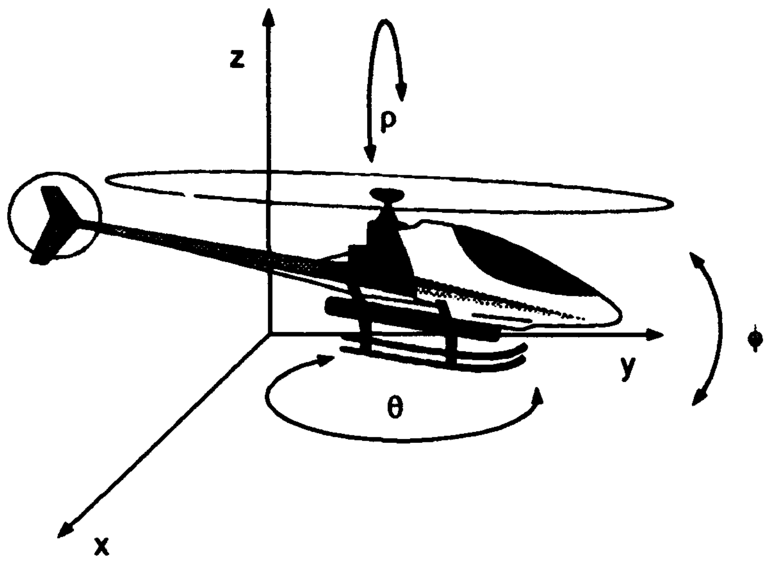
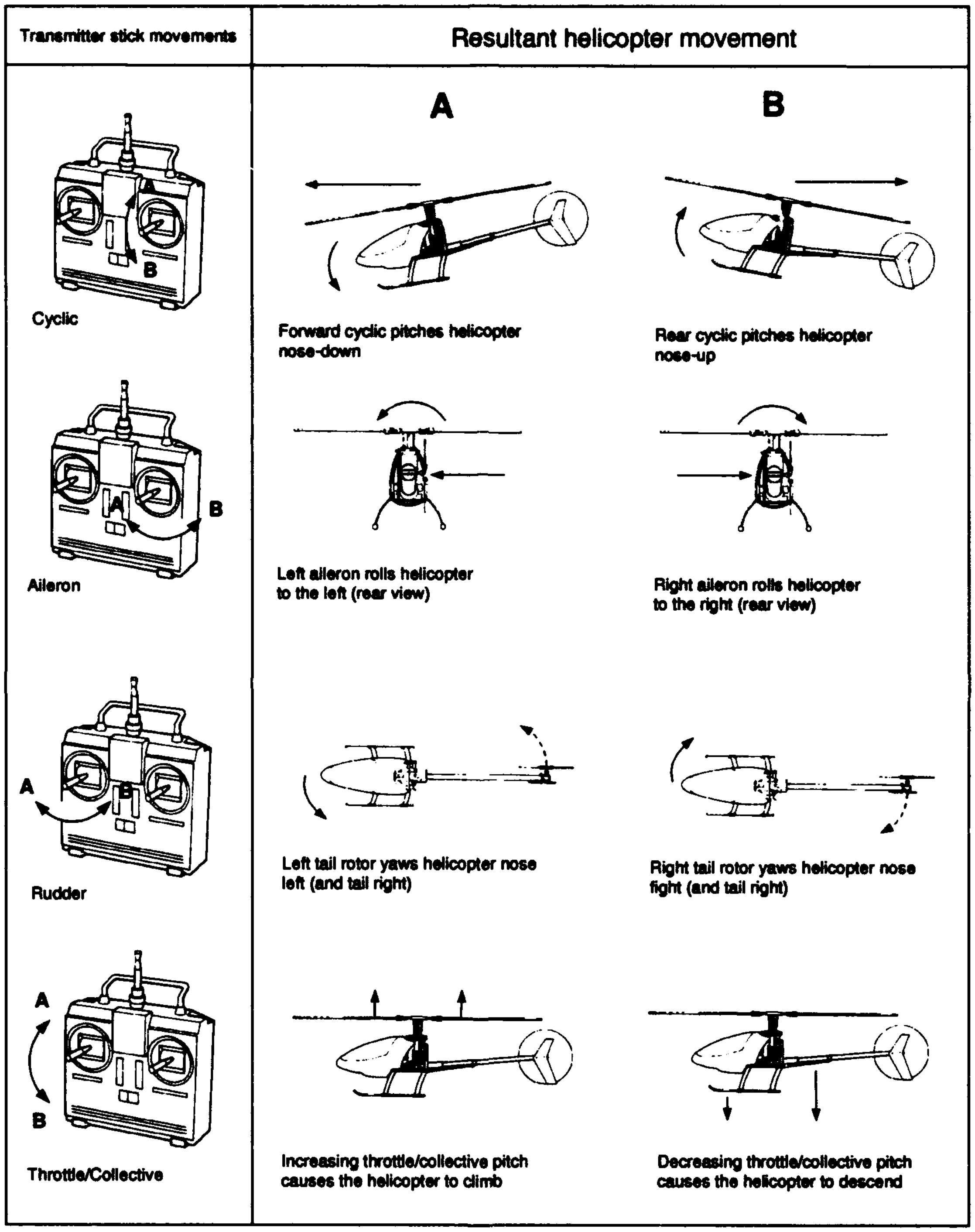


Figure 11. Helicopter Attitude and Position

3.5 Summary

The method of controller development devised by Nguyen can be directly applied to this particular control problem. The first and most important step is to train and test the emulator NN. The primary differences in the approach outlined here include the lack of necessity of BPTT and the lack of the plant dynamics for comparison during controller training.

By first decomposing the overall helicopter problem into a tail rotor emulator and control effort, the basic approach can be tested prior to attempting the more complicated task of emulating and eventually controlling the entire aircraft.


Transmitter stick movements
Resultant helicopter movement
A
B
Cyclic
Forward cyclic pitches helicopter nose-down
Rear cyclic pitches helicopter nose-up
Aileron
Left aileron rolls helicopter to the left (rear view)
Right aileron rolls helicopter to the right (rear view)
Rudder
Left tail rotor yaws helicopter nose left (and tail right)
Right tail rotor yaws helicopter nose fight (and tail right)
Throttle/Collective
Increasing throttle/collective pitch causes the helicopter to climb
Decreasing throttle/collective pitch causes the helicopter to descend

Please note that this is only a guide to control functions and is not a training procedure

Figure 12. Transmitter Layout and Contrc! Function (7:40)

IV. Results

This chapter details the results of the experimentation outlined in Chapter III, documenting the development of the emulator NN for the tail rotor and the helicopter through the use of the Nguyen method. This chapter will also discuss some preliminary work done in the development of the controller NN for the tail rotor.

4.1 Emulator

The emulator development of the tail rotor proved to be fairly straightforward. The various neural network models were prototyped and tested, varying the design parameters of each, such as number of hidden nodes, step size, and number of epochs. All resulted in accurate emulators of the tail rotor. The helicopter emulator was developed using the three-state model prototyped with the tail rotor. The results were equally as promising.

The emulator development consists of a two-phase process. Phase I concentrates only on the tail section, limiting movement to yaw (angular offset) only. Phase II then models the entire helicopter in a neural network.

4.1.1 Phase I. The primary step in developing a NN-based controller is to first design and test an emulator NN. In order to simplify the problem initially, this first attempt will only examine the tail section of the helicopter. This reduces the complexity and quantity of the input and output variables.

Using the FAME apparatus, 937 time samples of tail rotor performance were recorded for training and testing the emulator NN. Of these, the first 300 were used as training exemplars with the remaining to serve as the test exemplars.

Various input and output parameters were tested to obtain the best emulation of the tail rotor. The first configuration uses a three-state input vector, with the present and past two yaw positions as inputs and the next yaw position as the output. Also tested were four and five-state configurations, using the present yaw position along with the past three and four yaw positions. The last configuration tested uses the yaw position, velocity and acceleration as input vector and the next yaw position, velocity and acceleration as the output vector. These different configurations were tested, varying the number of hidden nodes, step size (η), and number of epochs.

4.1.2 Testing. In three-state configurations, the input vector to the net included the control vector (throttle and rudder) and the state vector (current and past two yaw positions), represented as $\mathbf{x} = [\mathbf{u}_k^T | \mathbf{s}_k^T]^T = [t,\ r |\ \theta_k,\ \theta_{k-1},\ \theta_{k-2}]^T$. The desired output of the net is the next yaw position (θ_{k+1}) (Figure 13).

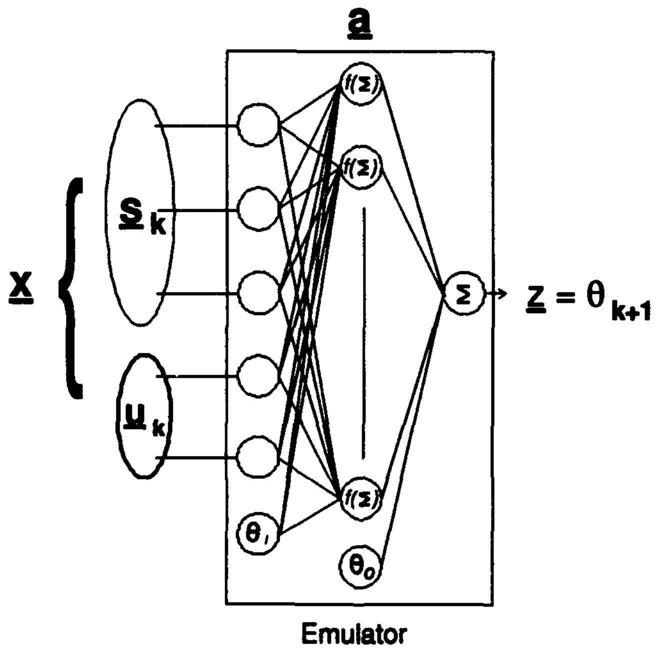


Figure 13. Emulator Neural Network

Using MATLAB® to prototype the NN and to run the backpropagation routine, the emulator turned out to be surprisingly accurate. After 10000 epochs over a training set of 300 exemplars, the emulator NN was tested over the entire data set of 900 exemplars. The NN tracked the output with amazing accuracy, considering over 600 of the exemplars were never presented for training. The final configuration used was a two-layer network with fifteen hidden nodes. The output of the network over the test set is shown in Figure 14.

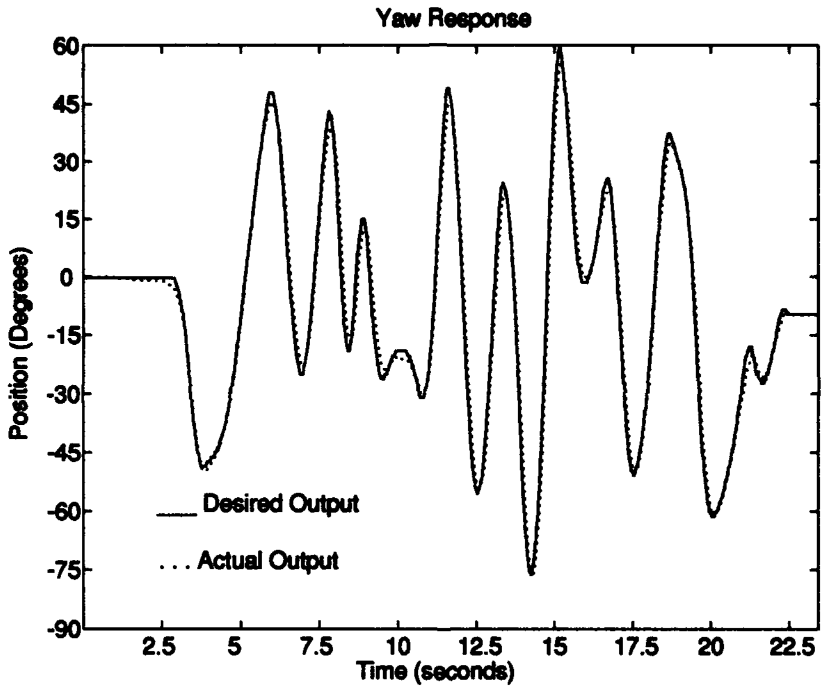


Figure 14. Tail Rotor Emulator Performance

This result in itself is remarkable. It shows a real, physical plant can be accurately modeled through use of a NN. Prior effort in developing emulator NNs used mathematical models of the plant. In this case, the actual plant was used to train the network.

The four and five-state configurations were also trained and tested in a similar manner. The assumption was more information about the past performance should produce a more accurate emulator. However, both of these configurations produced results very similar to the three-state configuration.

The emulator model using angular position, velocity, and acceleration was also prototyped and tested. Initially, the results of the velocity and acceleration model (v-a) appeared to indicate it was a better predictor. The yaw position was much more accurate that any of the delayed state models. (Figure 15).

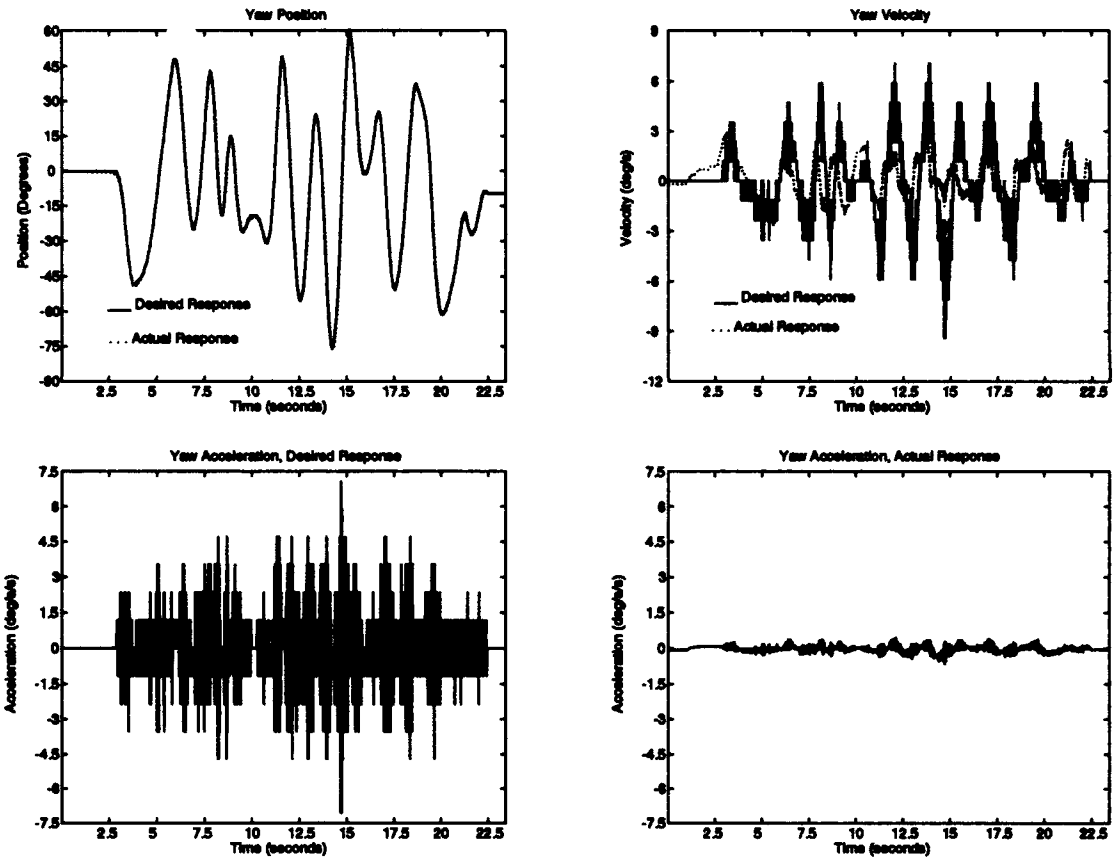


Figure 15. Tail Rotor Emulator NN Performance (Velocity-Acceleration Model)

When compared to the yaw position error of the three state configuration, the v-a model appear to have better performance at predicting the next yaw position (Figure 16). The mean square error is defined as in the backpropagation equations, i.e. $\frac{1}{2}||bfd - bfz||_2^2$.

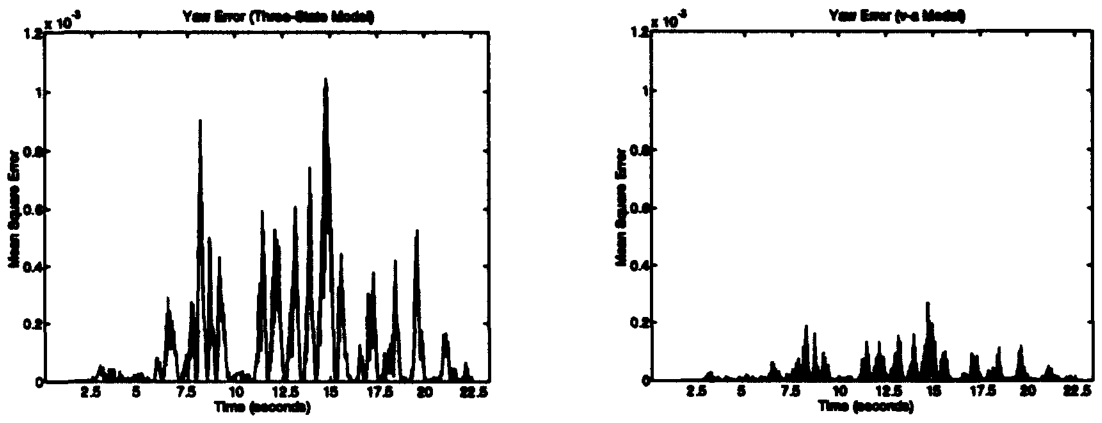


Figure 16. Comparison of position error of the three-state and velocity-acceleration models of the tail rotor emulator NN

At first it would appear the velocity-acceleration model performed much better than the three-state delay model. The next yaw position error is much lower for the velocity acceleration model. However, the next state vector of the v-a model consists not only of the yaw position, but also the velocity and acceleration at that point. When summed, the entire error is approximately the same as the three-state model (Figure 17).

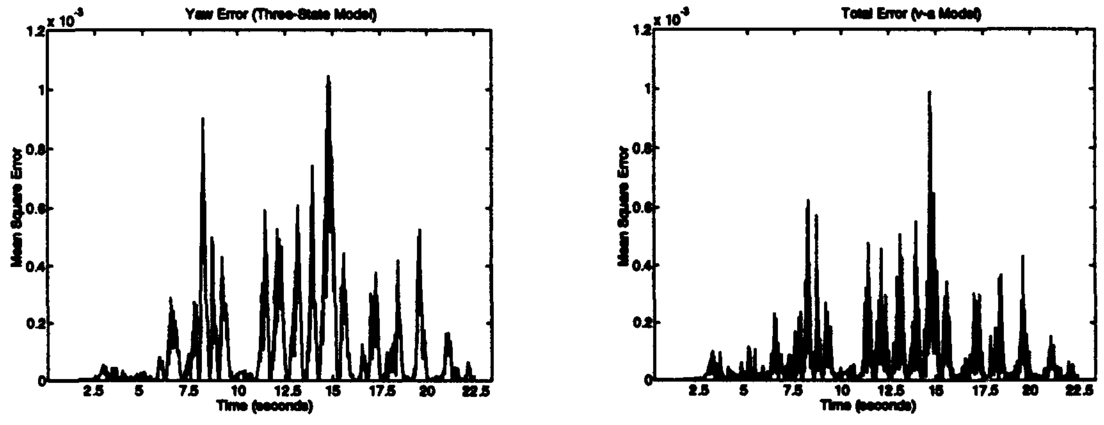


Figure 17. Comparison of the total error of the three-state and velocity-acceleration models of the tail rotor emulator NN

4.1.3 Phase II. The remainder of the helicopter control surfaces (aileron, cyclic, elevator) and spatial responses (pitch, roll, X-Y coordinates, altitude) are included in the plant emulator. Using the three-state delay configuration for the emulator. Again, the number of hidden nodes and step size were varied during testing.

Again, using the flight data recorded from the FAME apparatus, the exemplar set consists of a 500 sample segment from one set of data. The test set was 1131 data points from an entirely different flight.

The emulator results for the helicopter were equally as promising as the tail rotor (Figure 18). Most of the error appears to be concentrated primarily in the first and last one hundred or so samples of the test set (Figures 19 & 20). Since the helicopter is actually flying only during the middle portion of the test set, apparently the takeoff and landing performance is not fully parameterized in the NN.

4.2 Controller Results

Development of the controller proved to be more problematic. After developing the necessary MATLAB® program, the emulator was used to train the controller using the Nguyen method.

Initially, it appeared as if the controller was successfully trained. The error converged to an acceptably low level. However, subsequent test proved the tail rotor controller converged to a solution outside the range of the emulator input. Although the input data used for training the emulator was normalized, there was no way to insure the control output from the controller would be in the normalized range. Apparently, the controller found a solution outside the range of the emulator input values.

Thought was given to implementing a hard limiting function at the output of the controller, but this would not allow backpropagation due to the inability to differentiate this function. Another option was to include a sigmoid function at the output to the controller. However, it was felt this

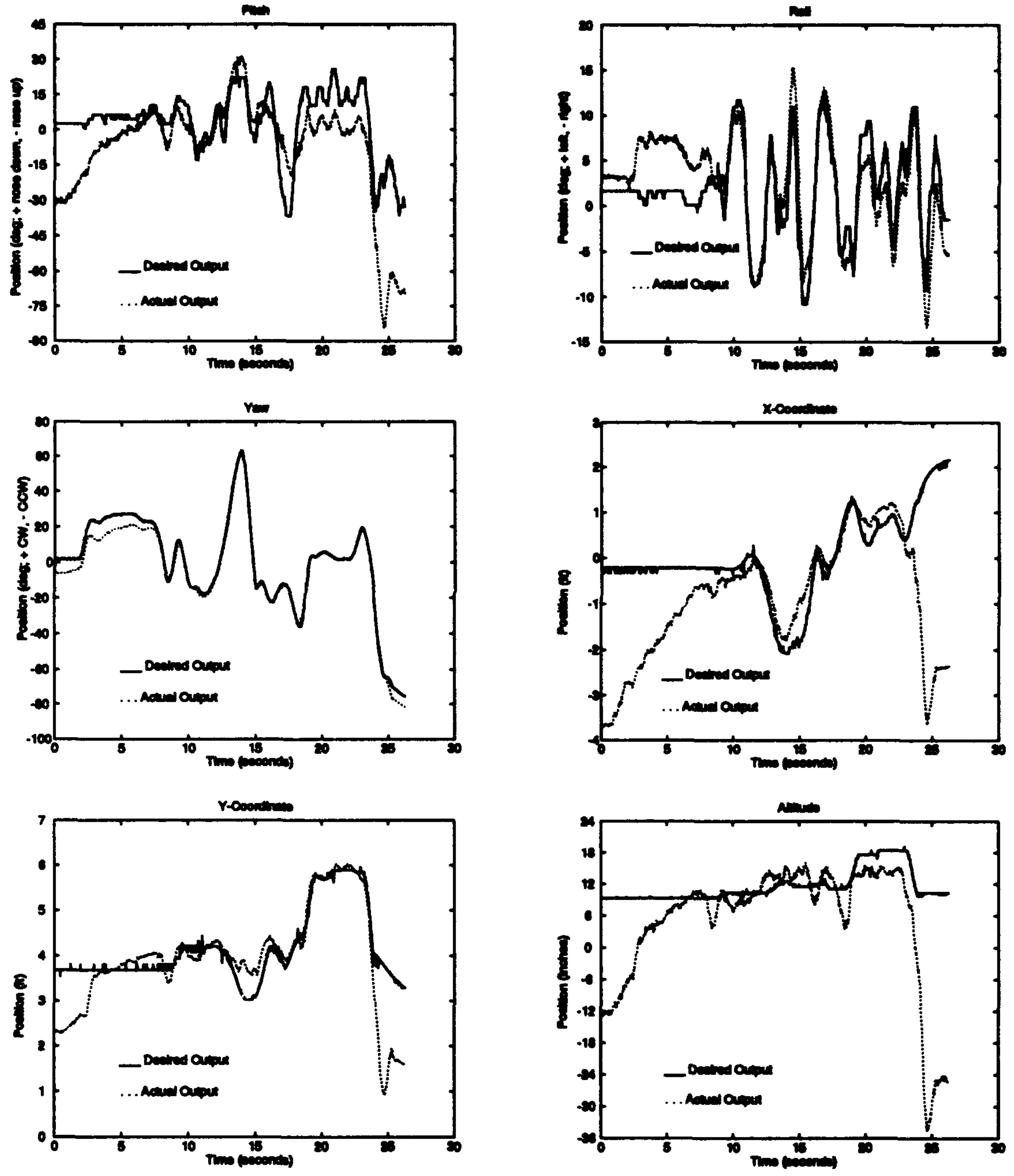


Figure 18. Helicopter Emulator NN Performance

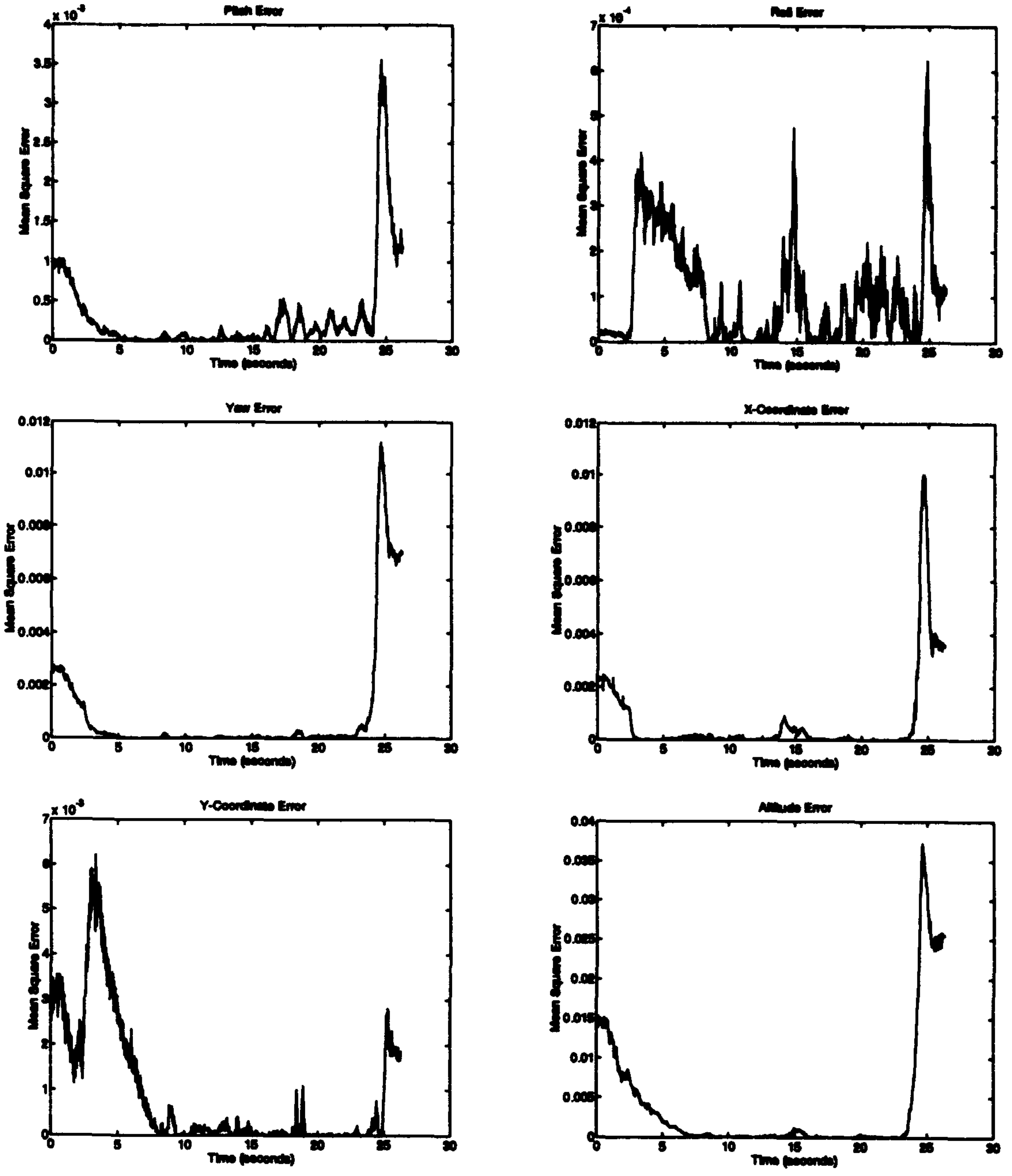


Figure 19. Helicopter Emulator NN Error

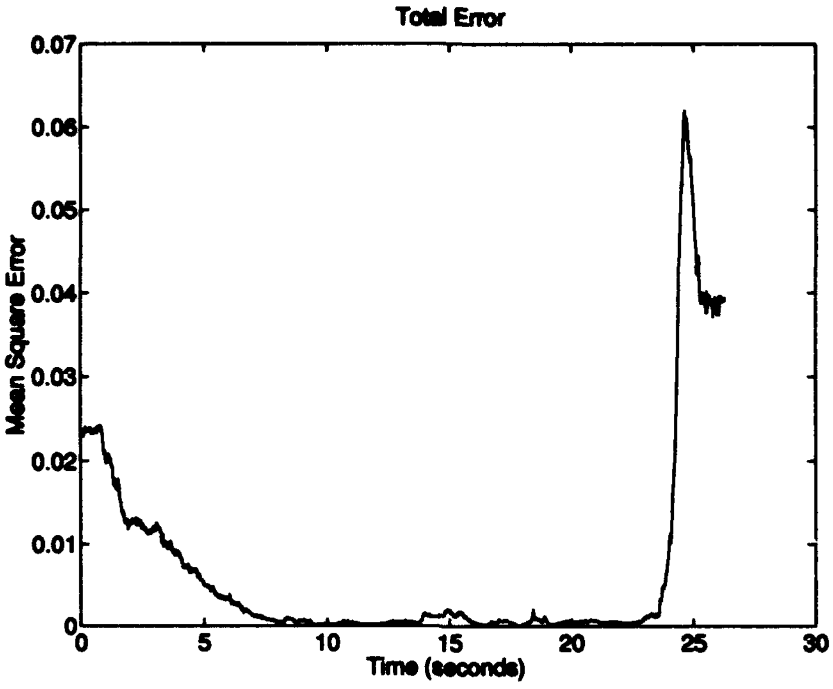


Figure 20. Helicopter Emulator NN Total Error

would substantially reduce the impact of error as it backpropagated through the system, thereby increasing training time and reducing accuracy.

Another possible solution was considered, but not investigated. If the control energy was included in the error vector, this would essentially ensure the control output would remain in an acceptable range for the emulator input. As the control energy increased, the error would also increase. Nguyen discusses this point and indicates it can be easily added incorporated into the backpropagation algorithm by adding the term $-\alpha u_k$ to the equivalent error (13:481). In this instance, u_k represents the control input at time k and α is scalar weighting factor.

4.3 Results

Artificial neural networks are capable of emulating dynamic non-linear plants with a high degree of accuracy. Both the tail rotor and the more complex component of the helicopter were reproduced by the NN. This is a crucial step in the development of a NN-based controller. These results also lead to the possible application of NNs to simulator development. The emulator NN eliminates the need to develop complex mathematical representations of the plant in order to simulate the plant. It only requires a sufficient type and quantity of examples of the plant's behavior.

V. Conclusions and Recommendations

5.1 Conclusions

The research conducted in the course of this thesis investigated the application of Nguyen self-learning neural network for control to the specific problem of controlling a scale-model helicopter. A critical requirement for the development of a control system is the successful emulation of the plant in a NN. This process involves the correct selection of input/output parameters necessary to correctly parameterize the plant.

Application of the method developed by Nguyen and Widrow was not applied to a real plant. By using simulation, the mathematical characterization of the plant was available when training the controller NN. The application of this method to a real plant does not permit the use of the plant dynamics when training the controller. Instead, the emulator NN alone must fully characterize the plant to be controlled.

5.2 Accomplishments

In this research, the following objectives have been successfully accomplished:

- It is possible for a neural network to accurately replicate the input-output mapping of a complex, nonlinear real plant. The emulator NN replicated the performance of the tail rotor and helicopter with a high degree of accuracy. This was accomplished without the development of mathematical models of the plant. The emulator is dependent only on the size of the net and training time. This ability to emulate a real plant has application not only to control problems but could also have application to simulator development.

- The application of neural networks to the control of a dynamic, nonlinear real plant has been investigated. Most previous efforts in NN-based control focused primarily on computer simulations of aircraft and other devices. The use of a real plant, although scale, is one step in the direction of application to full scale flight control.

- The Nguyen technique was applied to a problem involving a real plant. This method involves the use of two NNs and backpropagation through time (BPTT). The many vagaries in this article, including the memory intensive nature of BPTT, were clarified and documented.
- The Fast Adaptive Maneuvering Experiment apparatus was further developed and refined, thereby reducing the workload and flying expertise required by an experimenter. This refinement included numerous hardware and software modifications and development of documentation.
- MATLAB® proved to be a useful tool in NN research. All the equations derived in this research were implemented in MATLAB®. Its ability to handle vector and matrix representations permits rapid prototyping of NNs. The interpretive language allowed modifications to be quickly and easily implemented.

5.3 Recommendations

5.3.1 Controls Research. There remains a great deal of research remaining in the area of neural networks for controls. In general, investigation into the control systems using neural networks should continue. Specifically, research should continue to investigate Nguyen's approach of developing a controller neural network. This should include a duplication of his experiments, using both a computer simulation and an operational, scale-model tractor-trailer.

5.3.2 FAME. The Fast Adaptive Maneuvering Experiment is an excellent vehicle in the investigation of control application to real plants. This system permits the direct application of the plant dynamics rather relying on mathematical equations to model the system. Upgrades in the software and hardware of the FAME apparatus are continuously developed. GMU is currently developing a free-flying version of FAME. Also, advancing technology in R/C flying is creating the opportunity for more accurate measurement of flight dynamics. The latest breakthrough is a solid-state gyro system which is more precise by orders of magnitude than the present mechanical versions.

5.3.3 Simulators. This research has also uncovered the possible application of NNs to simulator development. The development of an emulator proves a complex, nonlinear plant can be represented in a NN. With this input-output mapping, it should be relatively straightforward task to develop a simulator based on a neural network.

Appendix A. Software, FAMEPC Version 6.0

Minor changes to version 6.0 of the software included changing the directory paths of the header files "pcdef.h" and "famedef.h". A small section of code was changed to select of COM 1 instead of COM 2 as the communications port interface to the MCU. Finally, the menu display was changed to reflect the correct software version and to correct a spelling error.

It is not mentioned in any of the FAME documentation, but it is also necessary to include a file called "generic.cal" in the same directory as FAMEPC.EXE. The program will run without it, but it will not report the state variables.

A change was made in the timing function "DelayUntil()" in "ControlLoop()." This function caused intermittent program lockup while attempting manually control the FAME helicopter. This function was replaced with the C-standard function "delay(10)", which produces a time delay of 10 milliseconds. Unfortunately, this causes inaccurate timing data to be recorded in "file1.trn." However, the data points are still recorded at even intervals of approximately 25 milliseconds.

```
/****************************************************************************/
/* File Name:  FAMEpc.C                                                     */
/*                                                                          */
/*  Authors:    Darrell Duane, Steve Suddarth and G-Z Sun                   */
/*  Update History:  Version 6.0, November 20, 1992                         */
/*                   Modified October 12, 1993 by Ronald E. Setzer          */
/*                                                                          */
/* *** indicates functions and their prototypes                             */
/* --- indicates ISRs                                                       */
/*                                                                          */
/****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <conio.h>
#include <float.h>
#include <ctype.h>
#include <bios.h>
#include <math.h>

#include "j:\workfile\borlandc\famedef.h"   /*file path to header files*/
#include "j:\workfile\borlandc\pcdef.h"


/* BEGIN PROGRAM:   BEGIN INITIALIZATION FUNCTIONS  */

/***************************************************************************/
/* Initializes ISR for TX & RX over serial port of PC                     */
/***************************************************************************/

void initializeISR(void)
{
 oldserialint = getvect(SERIALINT);   /* save the old ISR address */
 setvect(SERIALINT,newserialint);   /* attach the new ISR to the vector */
 outportb(MODEMCTL,(inportb(MODEMCTL) & 0xEF | DTR | RTS | OUT2));
 outportb(PIC01, (inportb(PIC01) & SERIALIRQ));  /* enable  the 8259 inter */
 outportb(PIC00,EOI);
 inportb(RXDATA);
 inportb(INTIDENT);
 inportb(LINESTATUS);
 inport(MODEMSTATUS);

/* printf(" Serial port initialized.\n"); */
}

/* Restore the old ISR attached to the com that we have used */

void RestoreOldISR(void)
{
  setvect(SERIALINT,oldserialint);
  printf("Old ISR restored  \n");
}


/**************************************************************************/
/* Function to initialize the UART, attach the new ISR, save the old ISR */
/**************************************************************************/

void InitSerialPort(void)
{
   initserial.serial_initial_bits.parity=  PARITY==2 ? 3 : PARITY ;
   initserial.serial_initial_bits.stopbits= STOPBITS-1;
   initserial.serial_initial_bits.wordlen= WORDLEN-5;
   initserial.serial_initial_bits.brk =0;
   initserial.serial_initial_bits.divlatch =1;
   outportb(LINECTL,initserial.serial_initial_char);
   outportb(DIVLSB,(char) ((115200L/BAUD) & 255));
   outportb(DIVMSB,(char) ((115200L/BAUD) >> 8));
   initserial.serial_initial_bits.divlatch = 0;
   outportb(LINECTL,initserial.serial_initial_char);
   initializeISR();
}

/****************************************************************************/
/* Initialized flags and semaphores for receiving data from HC11           */
/****************************************************************************/

void InitRXparm(void)
{
  RXstream=FALSE;
  RXindex=0;
  outportb(LINECTL,(inportb(LINECTL)|0x80));
/*  printf("DLAB bit in LCR is set = 1"); */
/*  printf("LCR = 0x%x\n",inportb(LINECTL)); */   /* Line Control Register */
/*  printf("BAUD0 = 0x%x ",inportb(DIVLSB)); */
/*  printf("BAUD1 = 0x%x ",inportb(DIVMSB)); */
  outportb(LINECTL,(inportb(LINECTL)&0x7f));
/*   printf("DLAB bit in LCR is set = 0 ");    */
/*   printf("DATA = 0x%x\n", inportb(RXDATA)); */  /* Receive data value */
/*   printf("LCR = 0x%x\n", inportb(LINECTL)); */  /* Line Control Register */
/*   printf("MCR = 0x%x\n", inportb(MODEMCTL)); */ /* Modem Control Register */
/*   printf("IER = 0x%x\n", inportb(INTENABLE)); */ /* Interrupt Enable
                                                     Register */
/*   printf("LSR = 0x%x\n", inportb(LINESTATUS)); */ /* Line Status
                                                      Register */
/*   printf("MSR = 0x%x\n", inportb(MODEMSTATUS)); */ /* Modem Status
                                                       Register Values */
/*   printf("IID = 0x%x\n", inportb(INTIDENT)); */ /* Interrupt Identification
                                                    & Causes */
Zero.Xcord = 0;
Zero.Ycord = 0;
Zero.Zcord = 0;
Zero.Pitch = 0;
Zero.Roll = 0;
Zero.Yaw = 0;
Zero.Gyro = 0;
} /* end intitalization of RX parameters */

/* END INITIALIZATION SECTION:    BEGIN ISR S */

/*-------------------------------------------------------------------*/
/* New communication interrupt service routine                       */
/*-------------------------------------------------------------------*/
static void interrupt far newserialint(void)
{ char identreg;
  identreg=inportb(INTIDENT);
  switch (identreg)
   {
   case 4:
 PC_RX_ISR();
 break;
   case 2:
 printf("TX ISR");
 break;
    default:
printf("default int\n");
inportb(RXDATA);
break;
    }
  outportb(PIC00,EOI);
}

/* END ISR'S:  BEGIN FUNCTIONS TX  */

/*----------------------------------------------------------------------*/
/* Initiates TX Sequence to the HC11                                    */
/************************************************************************/
void TXit(void)
{
char extra;
int i;

/* delay(delaytime);      */
EnablePC_RXint();
 while(getbit(inport(LINESTATUS),0))
   {    extra=inportb(RXDATA);
printf("Cleared byte = 0x%x = '%c' from serial port RX buffer.\n",
       (unsigned char)extra, extra);
    }
  while(getbit(inportb(LINESTATUS),5)==0) {putch('w'); putch('!'); }
   /* wait for TX to complete */
if(CommandChar>=REQ_DEFAULT_THROTTLE)
```

```
    {   outportb(TXDATA,CommandChar);
       /* printf("CharTXed: %u",CommandChar);   */

     TXtime=biostime(0,0);
        RXstream=TRUE;  /* it is expected to receive a stream */
        RXindex=(ELEVATOR-1);

     while(RXstream)
       {  if( (biostime(0,0) > (TXtime + (8*FIFTY_BIOS_MILLISECONDS))) )
  { printf("NAK \n");
    ClearWorkVar();
   }
       } /* end while RXstream */
   }
else  /* Transmit Servo Control String */
 {   RXstream=TRUE;  /* it is expected to receive a stream */
     RXindex=(ELEVATOR-1);
  for(i=ELEVATOR; i<RUDDER; i++)  /* Transmit Elevator thru Collective  */
     {
     RXchar=TRUE;
     outportb(TXDATA,ReqBuff[i]);
     TXtime=biostime(0,0);

 /*    delay(20);    REMOVE THIS DELAY WHEN RESTORING RX FUNCTION!!!  */
     while(RXchar)
 {  if( (biostime(0,0) > (TXtime + (8*FIFTY_BIOS_MILLISECONDS))) )
 { printf("NAK \n");
     ClearWorkVar();
  }
       } /* end while RXchar */
     } /* end for() */


    outportb(TXDATA,ReqBuff[RUDDER]);
       /* printf("CharTXed: %u",CommandChar);   */

     TXtime=biostime(0,0);

     while(RXstream)
       {  if( (biostime(0,0) > (TXtime +(8*FIFTY_BIOS_MILLISECONDS))))
 { printf("NAK \n");
     ClearWorkVar();
  }
       } /* end while RXstream */
   } /* end Else Transmit Servo Control String */
CommandChar=SERVO_CONTROL;
} /* end TXit */

/* END TX DATA:  BEGIN FUNCTIONS FOR RX DATA  */

/*-----------------------------------------------------------------------*/
```

```
/* Receiving from EVB ISR:  Ensures no errors then passes to CharRX()       */
/*--------------------------------------------------------------------------*/

void PC_RX_ISR(void)
{ WorkRXdata=inportb(RXDATA);
  /* putch(WorkRXdata); putch('*'); display values received from EVB */
  WorkLinestat=inportb(LINESTATUS);
if((getbit(WorkLinestat,3)!=0)||(getbit(WorkLinestat,2)!=0)||
   (getbit(WorkLinestat,1)!=0))
   { ClearWorkVar(); printf("RX ERROR:%u\n",(unsigned char)WorkLinestat); }
   /*  Error = TRUE */
  else
     CharRX();
} /* end RX ISR */


/*-------------------------------------------------------------------------*/
/* Function to receive data from the HC11                                  */
/*************************************************************************/
void CharRX(void)
{
  RXindex++;
/*  putch(WorkRXdata);  */

if(CommandChar>=REQ_DEFAULT_THROTTLE)
   switch(CommandChar)
     {
      case SER_REQ:
      case REQ_DEFAULT_THROTTLE:
      case REQ_DEFAULT_SER_VALS:
 if(RXindex < SER_ACK_STRING_LENGTH-1)
    AckBuff[RXindex]=WorkRXdata; /* put Servo Settings into array to be
                                        un-concatenated */
 else
    if(RXindex == SER_ACK_STRING_LENGTH-1)
{ if(WorkRXdata != Checksum(SER_ACK_STRING_LENGTH, AckBuff) )
                /* Checks if the checksum is correct */
     printf("Checksum Error: RXed Checksum= %x,
                            Calculated Checksum = %x \n",WorkRXdata,
                            Checksum(SER_ACK_STRING_LENGTH, AckBuff));
   else RXstream=FALSE;
}
      else { putch('s'); putch('!');  ClearWorkVar(); }
    break;

      case POT_REQ:
 if(RXindex < POT_ACK_STRING_LENGTH-1)
    AckBuff[RXindex]=WorkRXdata; /* put Potentiometer values into array to
                                        be un-concatenated */
 else
    if(RXindex == POT_ACK_STRING_LENGTH-1)
{ if(WorkRXdata != Checksum(POT_ACK_STRING_LENGTH, AckBuff) )
/* Checks if the checksum is correct */
     printf("Checksum Error: RXed Checksum= %x,
    Calculated Checksum = %x \n",WorkRXdata,
                             Checksum(POT_ACK_STRING_LENGTH, AckBuff));
  else RXstream=FALSE;
 }
      else { putch('p'); putch('!');  ClearWorkVar(); }
    break;

      default:
 printf("Unknown Command TXed to EVB requiring acknowledgement in CharRX:
                 %u\n",CommandChar);
 RXstream=FALSE;
 ClearWorkVar();
       break;

     } /* end switch() */
 else  /* POT ACK from Servo Control */
    {  if(RXindex < POT_ACK_STRING_LENGTH-1)
    AckBuff[RXindex]=WorkRXdata; /* put Potentiometer values into array to
                                              be un-concatenated */
 else
    if(RXindex == POT_ACK_STRING_LENGTH-1)
{ if(WorkRXdata != Checksum(POT_ACK_STRING_LENGTH, AckBuff) )
                /* Checks if the checksum is correct */
     printf("Checksum Error: RXed Checksum= %x,
                             Calculated Checksum = %x \n",WorkRXdata,
    Checksum(POT_ACK_STRING_LENGTH, AckBuff));
  else RXstream=FALSE;
 }
      else { putch('q'); putch('!');  ClearWorkVar(); }
     }  /* end Pot Ack from Servo Control */

RXchar=FALSE;

} /* end function CharRX */


/*-----------------------------------------------------------------------*/
/*  Initialize semaphores for RXing a new string from the HC11.          */
/*************************************************************************/
void ClearWorkVar(void)
{ int i;
  DisablePC_RXint();
  RXstream=FALSE;
  RXchar=FALSE;
  for(i=0; i<POT_ACK_STRING_LENGTH; i++) AckBuff[i]=0;
/*  printf("CWV\n"); */
}
```

```
/*  END TRANSFER FUNCTION OPERATIONS:  BEGIN POTENTIOMETER & POSITION DATA */

/*-----------------------------------------------------------------------------*/
/* Castes values from the RX buffer & displays them to the screen           */
/*****************************************************************************/
void DisplayPotData(unsigned char *raw_pot)
{
 clrscr();
 printf(" H     Pot =      %3d  \n", (int)raw_pot[H_POT_B]);
 printf(" Az    Pot =      %3d  \n", (int)raw_pot[AZ_POT_B]);
 printf(" El    Pot =      %3d  \n", (int)raw_pot[EL_POT_B]);
 printf(" Pitch Pot =      %3d  \n", (int)raw_pot[PITCH_POT_B]);
 printf(" Roll  Pot =      %3d  \n", (int)raw_pot[ROLL_POT_B]);
 printf(" Yaw   Pot =      %3d  \n", (int)raw_pot[YAW_POT_B]);
 printf(" Gyro =      %7d\n", ConcatInt(raw_pot[FROM_GYRO_MSB],
        raw_pot[FROM_GYRO_LSB]));
}

/*****************************************************************************/
/* Function that calculates the present state measured by the pots in rad   */
/*****************************************************************************/
void CalculateState(unsigned char *raw_vals, float *cal_state)
{
double Hangle, AZangle, ELangle, PitchAngle, RollAngle, YawAngle, GyroRaw;

/*  This raw inputs in radians */
 Hangle     = (Hslope * (double)((int)raw_vals[H_POT_B] - Hdco) );
 AZangle    = (AZslope * (double)((int)raw_vals[AZ_POT_B] - AZdco) );
 ELangle    = (ELslope * (double)((int)raw_vals[EL_POT_B] - ELdco) );
 PitchAngle = (PitchSlope * (double)((int)raw_vals[PITCH_POT_B] - PitchDCO) );
 RollAngle  = (RollSlope * (double)((int)raw_vals[ROLL_POT_B] - RollDCO) );
 YawAngle   = (YawSlope * (double)((int)raw_vals[YAW_POT_B] - YawDCO) );
 GyroRaw    = ConcatInt(raw_vals[FROM_GYRO_MSB], raw_vals[FROM_GYRO_LSB]);

cal_state[X_COORD] = (float)( (double)LENGTH_ARM1 * cos(Hangle)
     + (double)LENGTH_ARM2 * cos(Hangle + AZangle) * cos(ELangle) );
cal_state[Y_COORD] = (float)( (double)LENGTH_ARM1 * sin(Hangle)
     + (double)LENGTH_ARM2 * sin(Hangle + AZangle) * cos(ELangle) );
cal_state[ALTITUDE] = (float)( ((double)LENGTH_ARM2) * sin(ELangle) );
cal_state[PITCH] = (float)( RtoD * PitchAngle);
cal_state[ROLL]  = (float)( RtoD * RollAngle);
cal_state[YAW]   = (float)( RtoD * (Hangle + AZangle + YawAngle) );
cal_state[YAW_DOT]  = (float)(GyroRaw-GyroOffset);
}  /* end of function CalculateState */

void DisplayState(float *cal_state)
{
  int i;
  clrscr();
  for (i=0; i<NUM_SENSORS; i++)
    printf("%f             \n",cal_state[i]);
```

```
}

/*-----------------------------------------------------------------------*/
/* Loads calibration coefficients from generic.cal                        */
/*************************************************************************/
void LoadGenericCal(void)
{
cal = fopen("generic.cal","r");
 fscanf(cal,"%lf %d ",&Hslope, &Hdco);
 fscanf(cal,"%lf %d ",&AZslope, &AZdco);
 fscanf(cal,"%lf %d ",&ELslope, &ELdco);
 fscanf(cal,"%lf %d ",&PitchSlope, &PitchDCO);
 fscanf(cal,"%lf %d ",&RollSlope, &RollDCO);
 fscanf(cal,"%lf %d ",&YawSlope, &YawDCO);
 fscanf(cal,"%d ", &GyroOffset);
fclose(cal);
}

/*-----------------------------------------------------------------------*/
/* This function loads default values for the servos.                     */
/*************************************************************************/
void DefaultServoVals(void)
{
 printf("\nSending Default Servo Values \n");
   CommandChar=REQ_DEFAULT_SER_VALS;
   TXit();
    ServoVals[ELEVATOR]=  ELEVATOR_DEFAULT_B;
    ServoVals[AILERON]=   AILERON_DEFAULT_B;
    ServoVals[THROTTLE]=  THROTTLE_DEFAULT_B;
    ServoVals[COLLECTIVE]=COLLECTIVE_DEFAULT_B;
    ServoVals[RUDDER]=    RUDDER_DEFAULT_B;
} /* end function DefaultServoValues */

/***********************************************************************/
/*  Reads joystick once, sets servos to that value                    */
/***********************************************************************/
void SetServoFromStick(void)
{
    int i, jval;
    unsigned int joystick[5];

    readstick_array(joystick);
    for (i=0; i<NUM_SERVOS; i++)
      {
 if ((i==THROTTLE) || (i==COLLECTIVE))
jval = ( (((int) min_b_array[i])
   + conv_factor[i] * ((int)jmax[i] - (int)joystick[i])) );
 else
jval = ( (((int) min_b_array[i])
    + conv_factor[i] * ((int)joystick[i] - (int)jmin[i])) );
 jval = (jval < (int)max_b_array[i]  ? jval : max_b_array[i]);
 ReqBuff[i] = (unsigned char) (jval > (int)min_b_array[i]  ?
jval : min_b_array[i]);
      }
}

/****************************************************************************/
/*    ControlLoop takes generates one control vector and xmits it          */
/*    Also, it gets uncalibrated state, result is stored in AckBuff        */
/*    previous state is stored in OldAckBuff                               */
/****************************************************************************/

void ControlLoop(char ControlSwitch)
{
  int i;
  for (i=0; i<POT_ACK_STRING_LENGTH; i++)
OldAckBuff[i] = AckBuff[i];
  CommandChar = SERVO_CONTROL;
  if (ControlSwitch != '0')
    {
      switch(ControlSwitch)
{
    case 'J': SetServoFromStick(); break;
    default:  break;
}
/*    DelayUntil(LastDelayTime + LOOP_DELAY);  Original code commented out */
      delay(10);   /*delay 10 milliseconds; 12 Oct 93*/
    }
  else
    InitDelayTimer();

  ClearWorkVar();
  TXit();
}

/****************************************************************************/
/* Takes state and servo data and does what we tell it to do    */
/****************************************************************************/

void ProcessStateData(char DataSwitch, unsigned char *Servos,
unsigned char *UncalState)
{
  float CalState[NUM_SENSORS];
  switch(DataSwitch)
    {
      case 'U': DisplayPotData(UncalState);
break;
      case 'D': CalculateState(UncalState,CalState);
DisplayState(CalState);
break;
      case 'S': CalculateState(UncalState,CalState);
```

```
StoreState(Servos, CalState);
      case '0': break;
      default:  break;
    }
}

/***************************************************************************/
/* Takes input from joystick* to set values for servos iteratively      */
/***************************************************************************/

void DoControlLoop(char ControlSwitch, char DataSwitch,
char WriteControlWithNewState, char *InitMessage)
{   int i;
    printf("%s",InitMessage);

    ControlLoop('0');
    while(!kbhit())
      {  ControlLoop(ControlSwitch);
 if (WriteControlWithNewState)
ProcessStateData(DataSwitch, ReqBuff, AckBuff);
   else
ProcessStateData(DataSwitch, ReqBuff, OldAckBuff);
      }
    getch();
}    /* End DoControlLoop  */

/*  END SERVO CONTROL OPERATIONS:  BEGIN CALIBRATION COEFFICIENT OPERATIONS  */

void main()
{ unsigned char MenuChoice[3];
int ContinueIt;          /* used to allow for continuous operation */
  SerialPort=1;          /* set port to com 2; change to com 1, 12 Oct 93 */
  setup_comm(); /* set up communications */
  setup_arrays();

  ContinueIt=TRUE;
  LoadGenericCal();

while(ContinueIt)
  {
  printf("\n     Fast Adaptive Maneuvering Experiment -- PC Interface  Version
        6.0\n");
  printf("      By Darrell Duane, Steve Suddarth, G-Z Sun \n");
  printf("   (J)oystick control of servo values\n");
  printf("   (G)--control w/ joystick ... store state in file 'file1.trn'\n");
  printf("   (C)alibrate joystick \n");
  printf("   print calibrated (S)tate\n");
  printf("   (X) Repetitive potentiometer request (Q)uit \n\n");
  scanf("%1s", MenuChoice);
  clrscr();
  switch(toupper(MenuChoice[0]))
```

```
   {
   case 'J':
DoControlLoop('J','O',TRUE,"Under joystick control\n");
    /*  Written by SCS and G-ZS  */
     break;
   case 'G':
DoControlLoop('J','S',TRUE,"Closed-loop joystick\n");
WriteState("file1.trn");
ClearStateRecs();
break;
   case 'C':
cal_joystick();    /*  Written by SCS and G-ZS  */
     break;
   case 'Q':
RestoreOldISR();      /* Restore the old interrupt service routine */
ContinueIt=FALSE;
     break;

   case 'X':                  /*  Repetitvely request potentiometer value  */
DoControlLoop('O','U',TRUE,"Here's potentiometers, press any key\n");
     break;

   case 'S':                  /*  Repetitvely request calibrated state  */
DoControlLoop('O','D',TRUE,"Here's calibrated data, press any key\n");
     break;

   default:
       printf("Invalid key hit #1-- reenter \n\n");
     break;
    }   /* end switch() */
  } /* end while loop for ContinueIt == TRUE  */
}  /* end main routine */

/*  END MAIN FUNCTION:    BEGIN MISCELLANEOUS FUNCTIONS  */

/*  Initialize communications with the HC11  */

void setup_comm(void)
{
  disable(); /* General interrupt mask */
  DisablePC_TXint(); /* Local TX mask */
  clrscr();
  InitSerialPort(); /* Initialize the UART: baud, port...*/
  TXindex=0; /* Initialize the TX parameters */
  InitRXparm(); /* Initialize the RX parameters */
  DisablePC_RXint();
  enable();         /* General interrupt mask */
}

/*  Sets up arrays for indexing of various servos, etc.  */
```

```
void setup_arrays(void)
{
    int i;
    FILE *joy_cal_file;

    min_b_array[ELEVATOR] = ELEVATOR_LOWER_LIMIT_B;
    max_b_array[ELEVATOR] = ELEVATOR_UPPER_LIMIT_B;
    min_b_array[AILERON] = AILERON_LOWER_LIMIT_B;
    max_b_array[AILERON] = AILERON_UPPER_LIMIT_B;
    min_b_array[THROTTLE] = THROTTLE_LOWER_LIMIT_B;
    max_b_array[THROTTLE] = THROTTLE_UPPER_LIMIT_B;
    min_b_array[COLLECTIVE] = COLLECTIVE_LOWER_LIMIT_B;
    max_b_array[COLLECTIVE] = COLLECTIVE_UPPER_LIMIT_B;
    min_b_array[RUDDER] = RUDDER_LOWER_LIMIT_B;
    max_b_array[RUDDER] = RUDDER_UPPER_LIMIT_B;

    /* Read in joystick calibration */

    if ((joy_cal_file = fopen(JOY_CAL_FILE, "r")) != 0)
    /*added != 0 to remove warnings, 12 Oct 93*/
    {
      for (i=0; i<NUM_SERVOS; i++)
fscanf(joy_cal_file, "%d, %d, %f\n", &jmin[i], &jmax[i], &conv_factor[i]);
      fclose(joy_cal_file);
    }
}
/*-------------------------------------------------------------------------*/
/* Calculates checksum of sequences: ignores last char ie ignores checksum  */
/***************************************************************************/

unsigned char Checksum(int stringlength, unsigned char CheckBuff[])
{
unsigned char ChecksumResult = 0;
unsigned int sum             = 0;
int i;
 for(i=0; i < stringlength - 1; i++)
     sum+= CheckBuff[i];
 ChecksumResult = (unsigned char)sum;
 return ChecksumResult;
}  /* end checksum function */

/*-------------------------------------------------------------------------*/
/* concats 2 unsigned characters to an integer                             */
/***************************************************************************/

int ConcatInt(unsigned char MSbits,unsigned char LSbits)
{ int result;
   result=(unsigned int)MSbits;
   result=(result << 8);
   result=result+LSbits;
   return result;
}

/*-------------------------------------------------------------------------*/
/* Pauses for user to read message on screen.                             */
/***************************************************************************/

void WaitForEnter(void)
{
  printf(" Press any key to begin.\n");
    getch();
} /* end function WaitForEnter() */

/*-----------------------------------------------------------------------*/


/*  END MISCELLANEA  BEGIN JOYSTICK */

void readstick(state)
  stickstate *state;
  {
   unsigned int i, portval;

/*   delay(2);   REMOVED WHEN SYSTEM IS SLOWED DOWN */
   i= 0;
   state->lx = 0;
   state->ly = 0;
   state->rx = 0;
   state->ry = 0;
   asm cli; /* disable interrupts */
   outportb(GAMEPORT,0);
   while ( (portval = inportb(GAMEPORT) & 15)!=0 && i++<3500)
     {
       if (portval&1) (state->rx)++;
       if (portval&2) (state->ry)++;
       if (portval&4) (state->lx)++;
       if (portval&8) (state->ly)++;
     }
   asm sti; /* re-enable interrupts */
  }    /*  End readstick  */

void readstick_array(stick_array)
  unsigned int *stick_array;
{
stickstate state;
readstick(&state);
stick_array[ELEVATOR]   = state.ry;
stick_array[AILERON]    = state.rx;
stick_array[THROTTLE]   = state.ly;
stick_array[COLLECTIVE] = state.ly;
stick_array[RUDDER]     = state.lx;
}
```

```
void cal_joystick(void)
{   int i;
    unsigned int joystick[5];
    FILE *joy_cal_file;

    for (i=0; i<5; i++) {jmin[i] = 10000;  jmax[i] = 0;}
    printf("Move both joysticks full range to calibrate, then press key");
    while(!kbhit())
      {    readstick_array(joystick);
   for (i=0; i<NUM_SERVOS; i++)
     { if (joystick[i] < jmin[i]) jmin[i] = joystick[i];
       if (joystick[i] > jmax[i]) jmax[i] = joystick[i];  }
       }
    getch();

    /*  Find conversion factors from joystick to servos  */
    for (i=0; i<NUM_SERVOS; i++)
     { conv_factor[i] =
((float)(max_b_array[i] - min_b_array[i]) /
(float)(jmax[i] - jmin[i]));
     }
    joy_cal_file = fopen(JOY_CAL_FILE, "w");
    for (i=0; i<NUM_SERVOS; i++)
      fprintf(joy_cal_file, "%d, %d, %f\n", jmin[i], jmax[i],
              conv_factor[i]);
    fclose(joy_cal_file);
}

/****  Creates a state record and updates the FirstStateRec           ****/
/****  and LastStateRec pointers                                      ****/

StateRec *GimmeRec(void)
{   StateRec *ThisStateRec;
    if((ThisStateRec = calloc(1,sizeof(StateRec))) !=0)
    /*added != 0 to remove warnings, 12 Oct 93*/
      {
ThisStateRec->NextStateRec = 0L;

if (FirstStateRec!=0L)
    LastStateRec->NextStateRec = ThisStateRec;
  else
    FirstStateRec = ThisStateRec;

LastStateRec = ThisStateRec;
      }
    return(ThisStateRec);
}
char StoreState(unsigned char *ServoVals, float *CalState)
{   StateRec *StoreStateRec;
    int i;
    if ((StoreStateRec = GimmeRec()) != 0)/*added != 0, 12 Oct 93*/
      {
StoreStateRec->TimeInSeconds =
/*    (float)(LastDelayTime - FirstDelayTime)/(float)SECOND_TIME_COUNT;*/
   (float)(longtimer() - FirstDelayTime) / (float)SECOND_TIME_COUNT;
/*    LastDelayTime replaced with longtimer to record time increments  */
for(i=0; i<NUM_SERVOS; i++)
  (StoreStateRec->ServoVals)[i] = ServoVals[i];
for(i=0; i<NUM_SENSORS; i++)
  (StoreStateRec->CalState)[i] = CalState[i];
return(TRUE);
      }
      else
return(FALSE);
}

void WriteState(char *TheFile)
{  FILE *StateRecFile;
   StateRec *ThisStateRec;
   int i;
   if ((StateRecFile = fopen(TheFile, "w")) != 0)
     {
       ThisStateRec = FirstStateRec;
       while (ThisStateRec)
 {
    fprintf(StateRecFile,"%6.2f,   ",ThisStateRec->TimeInSeconds);
    for(i=0; i<NUM_SERVOS; i++)
      fprintf(StateRecFile, "%d, ",(ThisStateRec->ServoVals)[i]);
    fprintf(StateRecFile,"       ");
    for(i=0; i<NUM_SENSORS; i++)
      fprintf(StateRecFile, "%7.3f, ",(ThisStateRec->CalState)[i]);
    fprintf(StateRecFile,"\n");
    ThisStateRec = ThisStateRec->NextStateRec;
 }
     }
}

void ClearStateRecs(void)
  { StateRec *NextStateRec;
    while (FirstStateRec)
      {
NextStateRec = FirstStateRec->NextStateRec;
free(FirstStateRec);
FirstStateRec = NextStateRec;
      }
    FirstStateRec = 0L;
    LastStateRec = 0L;
  }

/******************************  TIMING STUFF  ************************/
```

```
#pragma inline
static void near dummy (void) {}
/*-----------------------------------------------------------------------*
Name            readtimer - read the complemented value of timer 0
Usage           unsigned readtimer (void);
(from Borland C Library)
*-----------------------------------------------------------------------*/

static unsigned near readtimer (void)
{
  asm pushf                    /* Save interrupt flag                    */
  asm cli                      /* Disable interrupts                     */
  asm mov  al,0h               /* Latch timer 0                          */
  asm out  43h,al
      dummy();                 /* Waste some time */
  asm in   al,40h              /* Counter --> bx                         */
  asm mov  bl,al               /* LSB in BL                              */
      dummy();                 /* Waste some time */
  asm in   al,40h
  asm mov  bh,al               /* MSB in BH                              */
  asm not  bx                  /* Need ascending counter                 */
  asm sti
  asm popf                     /* Restore interrupt flag                 */
  return( _BX );
}

unsigned long longtimer()
{  unsigned long btime;
   unsigned rtime;

   btime = biostime(0,0L);
   rtime = readtimer();
   if (btime!=biostime(0,0L))
return( (biostime(0,0L)<<16) | readtimer());
   else
return( (btime<<16) | rtime);
}

unsigned long LastDelayTime = 0L;
unsigned long FirstDelayTime = 0L;

void InitDelayTimer(void)
{  LastDelayTime = FirstDelayTime = longtimer();
}

/*The function DelayUntil() causes intermittent lockups of FAMEPC.  Replaced*/
/*with C function delay().  Requires changing LastDelayTime with longtimer()*/
/*so that timing data is recorded in file1.trn.  However, records time data */
/*recorded rather than time measured. R. Setzer, 12 Oct 93      */
```

```
unsigned DelayUntil(TargetTime)
  unsigned long TargetTime;
{
  if(   ( LastDelayTime=longtimer() )    > TargetTime) /* check for wraparound*/
    if (TargetTime-LastDelayTime > THIRTY_MINUTES_TIME)
      while( (LastDelayTime=longtimer()) > THIRTY_MINUTES_TIME);
    else
      return(1);                /* busted deadline */
  while( (LastDelayTime=longtimer()) < TargetTime)  ;      /* normal ops */
return(0);
}

unsigned long TimeSinceLastDelay(void)
{ return (longtimer() - LastDelayTime);
}

float MS_SinceLastDelay(void)
{ return (  (float)TimeSinceLastDelay() / (float)MILLISECOND_TIME_COUNT );
}

/* END PROGRAM */
```

```
/*-----------------------------------------------------------------------*/
/*                      George Mason University                          */
/*           Department of Electrical and Computer Engineering           */
/*                                                                       */
/* File name:  FAMEDEF.h                                                 */
/*                                                                       */
/*  Authors: Darrell Duane                                               */
/*  Update History:  Version 6.0, October ?, 1992                        */
/*                                                                       */
/*    Header file for FAME operating program in M68HC11                  */
/*                                                                       */
/*-----------------------------------------------------------------------*/

#ifndef FALSE
#define FALSE 0
#define TRUE !FALSE
#endif

/*-----------------------------------------------------------------------*/
/*      masks used for bitwise operations on registers or variables      */
/*-----------------------------------------------------------------------*/

#define MASK0 0xFE           /* 1111 1110 */
#define MASK1 0xFD           /* 1111 1101 */
#define MASK2 0xFB           /* 1111 1011 */
#define MASK3 0xF7           /* 1111 0111 */
#define MASK4 0xEF           /* 1110 1111 */
#define MASK5 0xDF           /* 1101 1111 */
#define MASK6 0xBF           /* 1011 1111 */
#define MASK7 0x7F           /* 0111 1111 */

#define CMASK0 ~MASK0        /* 0000 0001 */
#define CMASK1 ~MASK1        /* 0000 0010 */
#define CMASK2 ~MASK2        /* 0000 0100 */
#define CMASK3 ~MASK3        /* 0000 1000 */
#define CMASK4 ~MASK4        /* 0001 0000 */
#define CMASK5 ~MASK5        /* 0010 0000 */
#define CMASK6 ~MASK6        /* 0100 0000 */
#define CMASK7 ~MASK7        /* 1000 0000 */

#define IC1_NUM   0          /* number of input capture 1 */
#define IC2_NUM   1          /* number of input capture 2 */
#define IC3_NUM   2          /* number of input capture 3 */
#define NUM_IC    3          /* number of input captures in HC11  */

#define OC1_NUM   0          /* number of output compare 1 */
#define OC2_NUM   1          /* number of output compare 2 */
#define OC3_NUM   2          /* number of output compare 3 */
#define OC4_NUM   3          /* number of output compare 4 */
#define OC5_NUM   4          /* number of output compare 5 */
#define NUM_OC    5          /* number of output compares in HC11 */
```

```
#define PA3       3          /* pin number on port A     */
#define PA4       4          /* pin number on port A     */
#define PA5       5          /* pin number on port A     */
#define PA6       6          /* pin number on port A     */
#define PB0       0          /* pin number on port A     */
#define PB1       1          /* pin number on port A     */
#define PB2       2          /* pin number on port A     */

#define IC1RISE 0x10        /*bit pattern for IC1 to trigger interrupt on
                              rising edge */
#define IC1FALL 0x20       /*bit pattern for IC1 to trigger interrupt on
                             falling edge */


/*---------------------------------------------------------------------*/
/* defines for initializing RAM ISR jump table                          */
/*---------------------------------------------------------------------*/
#define JUMPEXTENDED 0x7E   /*Assembly language inst. for ISR jump table*/
#define VSCI    0x00C4      /* Serial Communications Interface         */
#define VSPI    0X00C7      /* Serial Peripheral Interface             */
#define VPAIE   0X00CA      /* Pulse Accumulator                       */
#define VPAO    0X00CD      /*                                         */
#define VTOF    0X00D0      /* Timer Overflow                          */
#define VTOC5   0X00D3      /* Output Compare 5                        */
#define VTOC4   0X00D6      /* Output Compare 4                        */
#define VTOC3   0X00D9      /* Output Compare 3                        */
#define VTOC2   0X00DC      /* Output Compare 2                        */
#define VTOC1   0X00DF      /* Output Compare 1                        */
#define VTIC3   0X00E2      /* Input Capture  3                        */
#define VTIC2   0X00E5      /* Input Capture  2                        */
#define VTIC1   0X00E8      /* Input Capture  1                        */
#define VRTI    0X00EB      /* Real Time Interrupt                     */
#define VIRQ    0X00EE      /* Maskable Interrupt Request              */
#define VXIRQ   0X00F1      /* Non-Maskable Interrupt Request          */
#define VSWI    0X00F4      /* Software Interrupt                      */
#define VILLOP  0X00F7      /* Illegal Operation                       */
#define VCOP    0X00FA      /* Computer Operating Properly             */
#define VCLM    0X00FD      /* Clock Monitor                           */
#define VRST    $E000       /* Restart Buffalo Monitor using assembly  */

/*---------------------------------------------------------------------*/
/*                  Definitions for Handshaking                         */
/*---------------------------------------------------------------------*/

/* Disable TX data buffer empty interrupt                               */
#define DisableTXbuffEmptyInt() ClearBit(SCCR2_Add,7)
/* Enable TX data buffer empty interrupt                                */
#define EnableTXbuffEmptyInt()  SetBit(SCCR2_Add,7)
/* Disable TX complete interrupt                                        */
#define DisableTXcompleteInt()  ClearBit(SCCR2_Add,6)
```

```
/* Enable TX complete interrupt                                              */
#define EnableTXcompleteInt()   SetBit(SCCR2_Add,6)
/* Disable RX start interrupt                                                */
#define DisableRXint()          ClearBit(SCCR2_Add,5)
/* Enable RX start interrupt                                                 */
#define EnableRXint()           SetBit(SCCR2_Add,5)
/* Disable idle line interrupt                                               */
#define DisableIdleInt()        ClearBit(SCCR2_Add,4)
/* Enable idle line interrupt                                                */
#define EnableIdleInt()         SetBit(SCCR2_Add,4)

#define SERVO_CONTROL 0

#define REQ_DEFAULT_THROTTLE 251
#define REQ_DEFAULT_SER_VALS 252

#define DUMMY 253

#define POT_REQ 254
#define SER_REQ 255

#define NUM_SERVOS 5
#define ZEROTH     0

#define ELEVATOR   0
#define AILERON    1
#define THROTTLE   2  /* index of servo control request chars          */
#define COLLECTIVE 3
#define RUDDER     4
#define TO_GYRO    5 /* 1.5 ms pulse always to gyro                          */
#define FROM_GYRO  6 /* input from gyro (YAW_DOT) */

#define NUM_PULSES 8                         /* number of pulses to send */

#define ELEVATOR_S   1
#define AILERON_S    2
#define TO_GYRO_S    3
#define THROTTLE_S   4  /* sequence of servo control output compares     */
#define COLLECTIVE_S 5
#define RUDDER_S     6
#define FROM_GYRO_S  7

#define AZ_POT_B         0    /* PE 4 */
#define ROLL_POT_B       1    /* PE 1 */
#define EL_POT_B         2    /* PE 5 */
#define YAW_POT_B        3    /* PE 2 */
#define H_POT_B          4    /* PE 6 */
#define PITCH_POT_B      5    /* PE 3 */
#define FROM_GYRO_MSB    6    /*PA 2 Used for setting DC offset of gyro time*/
#define FROM_GYRO_LSB    7
```

```
#define SER_ACK_STRING_LENGTH 6  /* 5 servo values & 1 Checksum       */
#define POT_ACK_STRING_LENGTH 9  /* 9 pot bytes = 8 bytes + 1 Checksum */

unsigned char ServoVals[NUM_SERVOS];

unsigned char AckBuff[POT_ACK_STRING_LENGTH]; /* TX Buffer for EVB, RX Buffer
                                                 for PC */
unsigned char ReqBuff[SER_ACK_STRING_LENGTH]; /* TX Buffer for PC */

int j;  /* delay char for TX */

/* index variables */
signed char RXindex;   /* index of chars RXed from PC */
signed char TXindex;   /* index of char to be TXed in the buffer         */
signed char TXend;     /* number of chars to TX */

/************************** EVB COM STATUS REGISTERS *********************/

unsigned char WorkSCSR;          /* status register of the SCI            */
unsigned char WorkRXdata;        /* work received data                    */

/* Sephamores                                                             */
/* TRUE if there is Noise,Framing error or an Overrun error               */
unsigned char NoiseFraming;
unsigned char Overrun;
unsigned char UnknownCommand; /* TRUE if an unknown command char is attempted
                                 to be TXed                  */
unsigned char OC3triggered;
unsigned char OC4triggered;
unsigned char OC5triggered;

/*--------------------------------------------------------------------*/
/*      Byte operation variables used to concatenate and cut bytes      */
/*--------------------------------------------------------------------*/
unsigned char LSBits;
unsigned char MSBits;          /* LSbits or MSbits to concat or results */
unsigned int  IntToSplit;
unsigned int  Concat2B;

/*  Global variables used to cut LongToSplit into 4 unsigned char    */
unsigned char Byte0;
unsigned char Byte1;
unsigned char Byte2;
unsigned char Byte3;
long LongToSplit;

double DoubleToSplit;

unsigned char CommandChar = 'x';  /* Temp til chksum and stop  */

/*--------------------------------------------------------------------*/
```

```
/*                        VARIABLES DEFINITION                              */
/*--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------*/
/*          VARIABLES RELATIVE TO THE INPUT CAPTURE FUNCTIONS              */
/*--------------------------------------------------------------------------*/
#define RX_RANK 2            /* Rank of the IC used for Receiver          */
#define RX_NUM (RX_RANK+1)   /* Number of the IC used for Receiver        */
#define APM_CHAN_RANK 2      /* rank of the pulse that gives the APM      */
#define TCNT_MAX_VAL 0xFFFF  /* Maximum value of the main 16 bit timer    */
#define OVERFLOW_MAX_VAL 0xFF/* Maximum value of the 8 bit overflow       */
     /* software counter (see type definition)   */
#define SET_PIN TRUE  /* used by to determine next state of output pin    */
#define CLEAR_PIN FALSE /* ditto */

/* 2 mhz, 500 ns per clock cycle of timer, so 2000 clock cycles = 1 ms    */
#define ONE_MS 2000   /* min acceptable time between two RX rising edges */
#define ONE_POINT_TWO_MS 2400
#define ONE_POINT_THREE_MS 2600
#define ONE_POINT_FIVE_MS  3000  /* Servo Midpoint                        */
#define ONE_POINT_SEVEN_MS 3400
#define ONE_POINT_EIGHT_MS 3600
#define ONE_POINT_NINE_MS 3800
#define TWO_MS 4000   /* max acceptable time between two RX rising edges */

#define ONE_MS_B 0   /* min acceptable time between two RX rising edges */
#define ONE_POINT_TWO_MS_B 50
#define ONE_POINT_THREE_MS_B 75
#define ONE_POINT_FIVE_MS_B  125  /* Servo Midpoint                        */
#define ONE_POINT_SEVEN_MS_B 175
#define ONE_POINT_EIGHT_MS_B 200
#define ONE_POINT_NINE_MS_B 225
#define TWO_MS_B 250   /* max acceptable time between two RX rising edges */

#define INTER_GROUP_DURATION 24000 /* coefficient used to keep low time
      consistent between each pulse.  This
      value is caluculated by
      (4000) * 6 = 24000. */
#define ELEVATOR_LOWER_LIMIT ONE_MS
#define ELEVATOR_UPPER_LIMIT TWO_MS
#define AILERON_LOWER_LIMIT ONE_MS
#define AILERON_UPPER_LIMIT TWO_MS
#define THROTTLE_LOWER_LIMIT ONE_MS /* should get green light             */
#define THROTTLE_UPPER_LIMIT TWO_MS /* should get red light               */
#define COLLECTIVE_LOWER_LIMIT ONE_POINT_TWO_MS   /* controls bind        */
#define COLLECTIVE_UPPER_LIMIT ONE_POINT_EIGHT_MS /* controls bind        */
#define RUDDER_LOWER_LIMIT ONE_MS
#define RUDDER_UPPER_LIMIT TWO_MS

#define ELEVATOR_LOWER_LIMIT_B ONE_MS_B
#define ELEVATOR_UPPER_LIMIT_B TWO_MS_B
#define AILERON_LOWER_LIMIT_B ONE_MS_B
```

```
#define AILERON_UPPER_LIMIT_B TWO_MS_B
#define THROTTLE_LOWER_LIMIT_B ONE_MS_B /* should get green light         */
#define THROTTLE_UPPER_LIMIT_B TWO_MS_B /* should get red light           */
#define COLLECTIVE_LOWER_LIMIT_B ONE_POINT_TWO_MS_B   /* controls bind    */
#define COLLECTIVE_UPPER_LIMIT_B ONE_POINT_EIGHT_MS_B /* controls bind    */
#define RUDDER_LOWER_LIMIT_B ONE_MS_B
#define RUDDER_UPPER_LIMIT_B TWO_MS_B

/* default servo values for initialization                               */
#define ELEVATOR_DEFAULT   ONE_POINT_FIVE_MS
#define AILERON_DEFAULT    ONE_POINT_FIVE_MS
#define THROTTLE_DEFAULT   THROTTLE_LOWER_LIMIT
#define COLLECTIVE_DEFAULT COLLECTIVE_LOWER_LIMIT
#define RUDDER_DEFAULT     ONE_POINT_FIVE_MS
#define TO_GYRO_DEFAULT    ONE_POINT_FIVE_MS

#define ELEVATOR_DEFAULT_B   ONE_POINT_FIVE_MS_B
#define AILERON_DEFAULT_B    ONE_POINT_FIVE_MS_B
#define THROTTLE_DEFAULT_B   THROTTLE_LOWER_LIMIT_B
#define COLLECTIVE_DEFAULT_B COLLECTIVE_LOWER_LIMIT_B
#define RUDDER_DEFAULT_B     ONE_POINT_FIVE_MS_B
#define TO_GYRO_DEFAULT_B    ONE_POINT_FIVE_MS_B


/*-----------------------------------------------------------------------*/
/*          6811 Evaluation Board Hardware Definitions                   */
/*-----------------------------------------------------------------------*/
#define LATCH_SCI 0x4000  /* address of flipflop to enable RX of data    */

/*-----------------------------------------------------------------------*/
/*          Stand hardware definitions                                   */
/*-----------------------------------------------------------------------*/
#define LENGTH_ARM1  63.5     /* length of lower arm in centimeters      */
#define LENGTH_ARM2  94.5     /* length of elevation arm in centimeters  */
/* pot number match A/D block numbers but do not match Port E pin numbers*/
#define AZ_POT          0       /* PE 4 */
#define ROLL_POT        1       /* PE 1 */
#define EL_POT          2       /* PE 5 */
#define YAW_POT         3       /* PE 2 */
#define H_POT           4       /* PE 6 */
#define PITCH_POT       5       /* PE 3 */
#define GYRO_CAL        6
#define SHOW_CAL_VALUE 7        /* used by FAMECAL.C to show cal values  */
#define EDIT_CAL_VALUE 8        /* used by FAMECAL.C to edit cal values  */
#define QUIT_VALUE      9       /* used by FAMECAL.C to quit             */

#define GYRO_CAL_COUNT 50       /* quantity of samples to take of gyro
                                   values */

/*-----------------------------------------------------------------------*/
/*          VARIABLES RELATIVE TO THE OUTPUT COMPARE FUNCTIONS           */
```

```
/*----------------------------------------------------------------------*/
/* Generates signal on a Port A pin & a Port B pin using one output     */
/* compare.                                                             */
/* arrays are larger than necessary so that output block numbers can be */
/* used as indices.                                                     */
/*----------------------------------------------------------------------*/
unsigned int TestServo;          /* Data RXed to be validated           */
unsigned int Thigh[NUM_PULSES]; /* Buffer to store time high for each pin*/
unsigned int LowerLimits[NUM_PULSES];  /* Lower limits for Servos       */
unsigned int UpperLimits[NUM_PULSES];  /* Upper limits for Servos       */

unsigned int ServoStatus;       /* indicates which servo is active      */

/*----------------------------------------------------------------------*/
/*          VARIABLES RELATIVE TO THE INPUT CAPTURE FUNCTIONS           */
/*----------------------------------------------------------------------*/
/* Awaits Rising or Falling edge on corresponding pin on port A and     */
/* triggers ISR upon reciept.                                           */
/*----------------------------------------------------------------------*/

unsigned int TimeAtRise_IC1; /* timer value when rise detected at PA2 */
unsigned int TimeAtFall_IC1;
/* used to calculate value for unsigned int YawDot, below */

/*----------------------------------------------------------------------*/
/*                            A/D CONVERSION                            */
/*----------------------------------------------------------------------*/

/* Bit patterns written to ADCTL to trigger A/D converters              */
#define PE0to3_ADCTL 0x10     /* Scan=off, Multiple channel,
 Convert Port E channels 0 through 3     */
#define PE4to7_ADCTL 0x14     /* Scan=off, Multiple channel,
 Convert Port E channels 4 through 7     */

#define PE0_ADCTL 0x00  /* Value to load ADCTL with to measure pin PE0   */
#define PE1_ADCTL 0x01  /* Value to load ADCTL with to measure pin PE1   */
#define PE2_ADCTL 0x02  /* Value to load ADCTL with to measure pin PE2   */
#define PE3_ADCTL 0x03  /* Value to load ADCTL with to measure pin PE3   */
#define PE4_ADCTL 0x04  /* Value to load ADCTL with to measure pin PE4   */
#define PE5_ADCTL 0x05  /* Value to load ADCTL with to measure pin PE5   */
#define PE6_ADCTL 0x06  /* Value to load ADCTL with to measure pin PE6   */
#define PE7_ADCTL 0x07  /* Value to load ADCTL with to measure pin PE7   */

/*----------------------------------------------------------------------*/
/*          variables relative to the position determination            */
/*----------------------------------------------------------------------*/
/* Stand potentiometer angles used to determine the Cartesian location   */
double Hangle, AZangle, ELangle, PitchAngle, RollAngle, YawAngle;

int Gyro;
```

```
/* Uncalibrated Stand potentiometer angles used to determine the Cartesian
   location   */
unsigned char Hraw, AZraw, ELraw, PitchRaw, RollRaw, YawRaw;

/* First Voltage from AD converter                                       */
unsigned char Hv0, AZv0, ELv0, PitchV0, RollV0, YawV0;

/* Second Voltage from AD converter                                      */
unsigned char Hv1, AZv1, ELv1, PitchV1, RollV1, YawV1;

#define DLY10  0x4E40      /* delay of 10 ms in term of main timer cycle */

/* Cartesian & Rotational position of the Helicopter                     */
int Xcord, Ycord, Zcord, Pitch, Roll, Yaw;

unsigned int YawDot;  /* difference between rise & fall times, above */

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
/*                         HC11 REGISTER VARIABLES                        */
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
unsigned int  *TCNT_Add;     /* main timer counter register              */
unsigned char *TMSK2_Add;    /* main timer interrupt mask                */
unsigned char *TFLG2_Add;    /* maintimer flag register                  */

unsigned int *IC_Add[NUM_IC]; /* pointer to input capture registers.    */
unsigned int *OC_Add[NUM_OC]; /* pointer to output compare registers    */

unsigned char *TMSK1_Add;   /* output compare and input capture int masks*/
unsigned char *TFLG1_Add;   /* output compare and input capture flags    */
unsigned char *TCTL2_Add;   /* input compare trigger type */
unsigned char *TCTL1_Add;   /* output compare automatic pin actions */
unsigned char *OC1D_Add;    /* output compare 1 control: data to set into
                               PAx */
unsigned char *OC1M_Add;    /* output compare 1 control: mask to set or not
                               PAx */
unsigned char *PACTL_Add;

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~ SCI REGISTERS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
unsigned char *SCSR_Add;        /* status register of the SCI:flags     */
unsigned char *SCDR_Add;        /* received and transmit data register  */
unsigned char *SCCR2_Add;       /* interrupt enables and state of SCI   */
unsigned char *SCCR1_Add;       /* data format 8 or 9 bits              */
unsigned char *BAUD_Add;        /* baud rate register                   */
unsigned char *LATCH_SCI_Add;   /* software controllable latch to connect
    pin PD0 to I/O connector            */

/*~~~ Port A & B registers: for sending pulses using output compare ~~~~~*/
unsigned char *PORTB_Add;
unsigned char *PORTA_Add;

/*~~~~~~~~~~~~~~~~~~~~~~~~~~ Port D registers  ~~~~~~~~~~~~~~~~~~~~~~~~~*/
```

```
unsigned char *PORTD_Add;        /* Port D                             */
unsigned char *DDRD_Add;         /* Data Direction for Port D          */
unsigned char *SPCR_Add;         /* SPI Control Register               */

/*-------------------------- A/D registers ---------------------------*/
unsigned char *OPTION_Add;      /* HC11 registers                      */
unsigned char *ADCTL_Add;       /* Control Register for A/D converter  */
unsigned char *ADR1_Add;        /* loc where converted values are stored */
unsigned char *ADR2_Add;
unsigned char *ADR3_Add;
unsigned char *ADR4_Add;

/*--------------------- EEPROM programming registers -----------------*/
unsigned char *PPROG_Add;        /* HC11 registers                     */
unsigned char *CONFIG_Add;

/*--------------------------------------------------------------------*/
/*  Declaration of the H11 register addresses defined in the library  */
/*  c:\introl\kjh\kjhstart.o11                                         */
/*--------------------------------------------------------------------*/
extern unsigned char H11PORTA;        /* i/o port A                    */
extern unsigned char H11PIOC;         /* parallel i/o control register */
extern unsigned char H11PORTC;        /* i/o port C                    */
extern unsigned char H11PORTB;        /* i/o port B                    */
extern unsigned char H11PORTCL;       /* alternate latch port C        */
extern unsigned char H11DDRC;         /* data direction for port C     */
extern unsigned char H11PORTD;        /* i/o port D                    */
extern unsigned char H11DDRD;         /* i/o data direction for port D */
extern unsigned char H11PORTE;        /* i/o port D                    */
extern unsigned char H11CFORC;        /* compare force register        */
extern unsigned char H11OC1M;         /* OC1 action mask register      */
extern unsigned char H11OC1D;         /* OC1 action data register      */

extern unsigned int H11TCNT;          /* timer counter register        */
extern unsigned int H11TIC1;          /* input capture register 1      */
extern unsigned int H11TIC2;          /* input capture register 2      */
extern unsigned int H11TIC3;          /* input capture register 3      */
extern unsigned int H11TOC1;          /* output compare register 1     */
extern unsigned int H11TOC2;          /* output compare register 2     */
extern unsigned int H11TOC3;          /* output compare register 3     */
extern unsigned int H11TOC4;          /* output compare register 4     */
extern unsigned int H11TOC5;          /* output compare register 5     */

extern unsigned char H11TCTL1;        /* timer control register 1      */
extern unsigned char H11TCTL2;        /* timer control register 2      */
extern unsigned char H11TMSK1;        /* main timer interrupt mask 1   */
extern unsigned char H11TFLG1;        /* main timer interrupt flag 1   */
extern unsigned char H11TMSK2;        /* main timer interrupt mask 2   */
extern unsigned char H11TFLG2;        /* misc timer interrupt flag 2   */
extern unsigned char H11PACTL;        /* pulse acc control register    */
extern unsigned char H11PACNT;        /* pulse acc count register      */
extern unsigned char H11SPCR;        /* SPI control register         */
extern unsigned char H11SPSR;        /* SPI status register          */
extern unsigned char H11SPDR;        /* SPI data in/out              */
extern unsigned char H11BAUD;         /* SCI baud rate control        */
extern unsigned char H11SCCR1;        /* SCI control register 1       */
extern unsigned char H11SCCR2;        /* SCI control register 2       */
extern unsigned char H11SCSR;         /* SCI status register          */
extern unsigned char H11SCDR;         /* SCI data                     */
extern unsigned char H11ADCTL;        /* A to D control register      */
extern unsigned char H11ADR1;         /* A to D result 1              */
extern unsigned char H11ADR2;         /* A to D result 2              */
extern unsigned char H11ADR3;         /* A to D result 3              */
extern unsigned char H11ADR4;         /* A to D result 4              */
extern unsigned char H11OPTION;       /* System configuration options */
extern unsigned char H11COPRST;       /* arm /reset COPtimer circutry */
extern unsigned char H11PPROG;        /* EEPROM programming control   */
extern unsigned char H11HPRIO;        /* highest priority I bit and misc*/
extern unsigned char H11INIT;         /* RAM /io mapping register     */
extern unsigned char H11TEST1;        /* factory test control         */
       /* COP, ROM, &EEPROM enables     */
extern unsigned char H11CONFIG;

#define ACIASR_ADDRESS 0x9800;        /*  ACIA status register          */
#define ACIADR_ADDRESS 0x9801;        /*  ACIA data register            */

unsigned char *ACIASR_Add;
unsigned char *ACIADR_Add;

unsigned char getIt;   /* Bytes where data is RXed */


/**************************************************************************/
/*                             PROTOTYPES                                 */
/**************************************************************************/
/* functions for measuring pots and calculating positions and angles     */

/* FAMEINIT.c:  FAME Project General Initialization Functions */

void InitConstantVariables(void);
void InitPointer(void);
void InitADconverter(void);

/* FAMEINI2.c:  FAMEMAIN Specific Initialization Functions */

void InitVectorTable(void);
void InitServos(void);
void SCI_init_TX(void);   /* initializes TX to PC                        */
void SCI_init_RX(void);   /* initialize for reception over SCI           */
void ACIA_init(void);
void InitOC(int OCnum, int Enable);
void InitICs(void);
```

```
/* FAMEISR.C   ISRs */

void OC5_ISR(void);
void IC1_ISR(void);

/* FAMEMAIN.C:  Transmission / Acknowlegement prototypes              */

void DecodeAndStoreServoByte(void);
void FillPotAckBuff(void);
void FillSerAckBuff(void);
void TXackBuff(void);
void ProcessWorkRXdata(void);
/* int main(int); */
void putByte(unsigned char putIt);  /* TXes byte to PC:  Byte TXed is found
                                       in putIt      */

/* FAMELIB.C: functions for doing basic bit operations on register settings*/

void SplitInt(int i);   /* function that calls asm function below       */
int ConcatInt(unsigned char MSbits,unsigned char LSbits);
unsigned char Checksum(int maxnum, unsigned char CheckArray[]);
void ClearBit(unsigned char *pointer,int NumBit);
void ClearFlag(unsigned char *pointer,int NumBit);
void SetBit(unsigned char *pointer,int NumBit);
unsigned char GetBit(unsigned char *pointer,int NumBit);
unsigned char GetBitChar(unsigned char reg,int NumBit);

/* ASSEMBLY Routines */
void splitInt(void);   /* assy lang to prepare pos values for TX to PC */
void concatInt(void);
void getByte(void);    /* loops until Byte is RXed:  Byte is placed in getIt*/
```

```
/***************************************************************************/
/* Filename: PCdef.h  Ver 5.0          , 1992                             */
/*                                                                         */
/*                      definitions for FAMEPC.c                          */
/***************************************************************************/

/****************************** SERVOS ************************************/
unsigned char ServoInc[NUM_SERVOS];
unsigned char EVBservoVals[NUM_SERVOS];
void DefaultServoVals(void);
void VariableServoVals(void);
void UserEnteredServoVals(void);
void DisplayServoVals(void);
void DisplayEVBservoVals(void);
void EqualizeServoVals(void);

/************************** POTENTIOMETERS ********************************/

int GyroRaw;    /* raw value of gyro */

/**************************** CALIBRATION *********************************/

#define PI  3.14159265359    /* value for pi */
double PiOverTwo=1.5707963268;  /* compute the constant for later use    */
double PiOverFour=0.7853981634; /* compute the constant for later use    */
#define RADIANS_TO_DEGREES 57.2957795131  /*conversion  */
#define DEGREES_TO_RADIANS 0.0174532925199
double RtoD = RADIANS_TO_DEGREES;
float DtoR = DEGREES_TO_RADIANS;

#define H_ANGLE0 0          /* calibration location 0 for H pot */
#define H_ANGLE1 (PI)
#define EL_ANGLE0 0
#define EL_ANGLE1 (-PiOverFour)
#define AZ_ANGLE0 0
#define AZ_ANGLE1 (PiOverTwo)
#define Pitch_ANGLE0 0
#define Pitch_ANGLE1 N/A   /* this value prompted for from user */
#define Roll_ANGLE0 0
#define Roll_ANGLE1 N/A   /* this value prompted for from user */
#define Yaw_ANGLE0 0
#define Yaw_ANGLE1 (PiOverTwo)
#define ATOD_ERROR_LIMIT 10  /* sum of max differences in four A/D samples */

double Hslope, AZslope, ELslope, PitchSlope, RollSlope, YawSlope;
int Hdco, AZdco, ELdco, PitchDCO, RollDCO, YawDCO, GyroOffset; /*DC offsets*/

int PotChoice;  /* Choice of potentiometer to calibrate */
int PrintValue; /* Value to print to screen when using Defines */

FILE *cal;  /* file pointer for calibration coefficients */
```

```
void LoadCalVals(void);
void SaveCalVals(void);
void DisplayCalVals(void);
void PrintPotOptions(void);
void PrintOtherOptions(void);
unsigned char MeasurePot(void);
void EditCoefficients(void);
void InternalCal(void);
void CalibrateAngles(void);

/****************************** POSITION **********************************/

void DisplayCalPotData(void);


/*************************** KEYBOARD CONTROL ****************************/

int ContinueKeyboard; /* indicates whether PC should continue accepting
                         keystrokes to vary servo control values. */
void KeyboardServoVals(void);


/**************************** DYNAMIC CONTROL ****************************/

int ContinueControl;
int UpdateStep;     /* used to indicate point at which function is in for
                       varying transfer values real time */

typedef struct StateDummy{
   float Xcord,
 Ycord,
 Zcord,
 Pitch,
 Roll,
 Yaw,
 Gyro; } STATE;

STATE Zero;  /* Vector of Zeros */

STATE ElevatorDoub,AileronDoub,ThrottleDoub,CollectiveDoub,RudderDoub;
STATE ElevatorInt,AileronInt,ThrottleInt,CollectiveInt,RudderInt;
STATE ElevatorProp,AileronProp,ThrottleProp,CollectiveProp,RudderProp;

STATE ElevatorDoubCr,AileronDoubCr,ThrottleDoubCr,CollectiveDoubCr,RudderDoubCr;
STATE ElevatorIntCr,AileronIntCr,ThrottleIntCr,CollectiveIntCr,RudderIntCr;
STATE ElevatorPropCr,AileronPropCr,ThrottlePropCr,CollectivePropCr,RudderPropCr;
STATE Doub,Int,Prop;
STATE Present,Desired;
STATE ErrorMinusOne,ErrorMinusTwo,ErrorMinusThree,ErrorMinusFour;
                                                    /* Prop serves as Error */
```

```
STATE *ToChangeDoub, *ToChangeInt, *ToChangeProp;
STATE *ChangeRateDoub, *ChangeRateInt, *ChangeRateProp;

FILE *Transfer;
void LoadTransferVals(void);  /* also loads Change rates */
void SaveTransferVals(void);  /* also saves Change rates */
void DisplayTransferVals(void);
void DisplayChangeRates(void);
void Dtv(STATE *DisplayIt);
void DisplayVector(STATE *todisp);
void DisplayError(void);
void EditTransferVals(void);
void Edit(STATE *toedit);
void UpdateTransferVals(void);
void CopyState(STATE *Copy, STATE *Original);
void Add(STATE *Equals, STATE *First, STATE *Second);
void Subtract(STATE *Equals, STATE *First, STATE *Second);
void SubtractRot(STATE *Equals, STATE *First, STATE *Second);
void CalculateState(unsigned char *raw_vals, float *cal_state);
void DisplayState(float *cal_state);
signed char SumItAll(STATE *Doub, STATE *ServoDoub, STATE *Int, STATE *ServoInt,
                     STATE *Prop, STATE *ServoProp);
signed char SumItAllPrint(STATE *Doub, STATE *ServoDoub, STATE *Int,
                          STATE *ServoInt, STATE *Prop, STATE *ServoProp);
void SaveDesiredCalPotData(void);
void DynamicControl(void);
void PDServoVals(void);

/********************************** TX ************************************/


/*---------------------Serial Port Addresses-----------------------------*/
/*    8250 UART  base port  Address: COM1=0x3f8 , COM2=0x2f8            */
/*---------------------------------------------------------------------*/
int SerialPort=1;      /* COM1 or COM2  */

#define BASE          (0x3f8-((SerialPort-1)<<8))
#define TXDATA         BASE        /*  transmit data */
#define RXDATA         BASE        /*  receive data  */
#define DIVLSB         BASE        /*  baud rate divisor lsb */
#define DIVMSB        (BASE+1)     /*  baud rate divisor msb */
#define INTENABLE     (BASE+1)     /*  interrupt enable */
#define INTIDENT      (BASE+2)     /*  interrupt ident'n */
#define LINECTL       (BASE+3)     /*  line control   */
#define MODEMCTL      (BASE+4)     /*  modem control  */
#define LINESTATUS    (BASE+5)     /*  line status    */
#define MODEMSTATUS   (BASE+6)     /*   modem status   */

/* -------------- serial interrupt values --------------------------------*/

#define IRQ  (4-(SerialPort-1))       /* 0-7 =IRQ0 - IRQ7 */
```

```
#define SERIALINT  (12 -(SerialPort-1))    /* interrupt  vector */
#define SERIALIRQ  (~(1 << IRQ))
#define PIC01  0x21   /* 8259 programmable interrupt controller  */
#define PIC00  0x20   /* "         "             " "        */
#define EOI    0x20   /*  End of interrupt command  */

/*----------------Modem control register -----------------------------------*/
#define DTR    1
#define RTS    2
#define OUT2   8

int PARITY     =0;      /* parity  none  */
int STOPBITS   =1;      /* 1 stopbit  or 2 */
int WORDLEN    =8;      /* length of the data or 7 */
int BAUD       =19200;

/*------------ serial port initialization parameter byte -------------------*/

static union {
   struct{
unsigned wordlen  : 2;
unsigned stopbits : 1;
unsigned parity   : 3;
unsigned brk      : 1;
unsigned divlatch : 1;
    } serial_initial_bits;
   char serial_initial_char;
} initserial;

/* Local mask to enable,disable TX interrupts */

#define EnablePC_TXint()   outportb(INTENABLE,setbit(inport(INTENABLE),1))
#define DisablePC_TXint()  outportb(INTENABLE,clearbit(inportb(INTENABLE),1))
#define ONE_BIOS_SECOND 20
#define ONE_QUARTER_BIOS_SECOND 5
#define FIFTY_BIOS_MILLISECONDS 1
long int TXtime;  /* time values are TXed */

static void (interrupt far *oldserialint)(void);
static void interrupt far newserialint(void);
void InitSerialPort(void);
void initializeISR(void);
int readserial(void);
int writeserial(void);
void clear_serial_queue(void);
void RestoreOldISR(void);
void TXit(void);

/********************************* RX ************************************/

#define EnablePC_RXint()   outportb(INTENABLE,setbit(inportb(INTENABLE),0))
```

```
#define DisablePC_RXint()  outportb(INTENABLE,clearbit(inportb(INTENABLE),0))
#define RXint_Enabled getbit(inportb(INTENABLE),0)

int RXstream;        /* Semaphore to indicate Stream being RXed */
int RXchar;          /* Semaphore to indicate Char RXed */

unsigned char WorkLinestat;        /* reception linestatus */
unsigned char RXcomplete;  /* sephamore set when the stream is correct */
unsigned char NewData;   /* sephamore set when new values are RXed */

void InitRXparm(void);
void ClearWorkVar(void);
unsigned char Checksum(int maxnum, unsigned char CheckArray[]);
void CharRX(void);
void PC_RX_ISR(void);

/********************************* MISC *************************************/
#define SOUNDDELAY 4
FILE *fopen();
int fclose();
void WaitForEnter(void);
int power(int a, int b);
long ConcatLong(unsigned char Byte0, unsigned char Byte1, unsigned char Byte2,
                unsigned char Byte3);
double ConcatLongD(unsigned char Byte0, unsigned char Byte1, unsigned char Byte2,
                   unsigned char Byte3);


/* Communication Prototypes  */

void LoadGenericCal(void);

/* Basic prototypes  */

void setup_comm(void);
void setup_arrays(void);

/* Joystick defines and prototypes */

#define GAMEPORT 0x203
typedef struct {unsigned int lx, ly, rx, ry;} stickstate;
void readstick(stickstate *);
void readstick_array(unsigned int *);
void cal_joystick(void);
#define JOY_CAL_FILE "joycal.0"


float conv_factor[5];
unsigned int jmin[5], jmax[5];     /*  Joystick calibration  */

/*  miscellaneous changes  */
```

```
unsigned char OldAckBuff[POT_ACK_STRING_LENGTH]; /* stores one history */

void SetServoFromStick(void);
void ControlLoop(char ControlSwitch);
void ProcessStateData(char DataSwitch, unsigned char *Servos,
unsigned char *UncalState);
void DoControlAndDataLoop(char ControlSwitch, char DataSwitch,
char WriteControlWithNewState, char *InitMessage);

unsigned PreviousControlTime;      /* Keeps track of when last control
cmd was sent */


void DisplayPotData(unsigned char *raw_pot);

int delaytime = 0;  /* time to delay between TX */


#define NUM_SENSORS  7

#define PITCH 0
#define ROLL 1
#define ALTITUDE  2
#define Y_COORD 3
#define YAW 4
#define X_COORD 5
#define YAW_DOT 6

float state[NUM_SENSORS];

typedef struct state_rec {
float TimeInSeconds;
unsigned char ServoVals[NUM_SERVOS];
float CalState[NUM_SENSORS];
struct state_rec *NextStateRec;
}
StateRec;

StateRec *FirstStateRec = 0L;
StateRec *LastStateRec = 0L;

StateRec *GimmeRec(void);
char StoreState(unsigned char *ServoVals, float *CalState);
void WriteState(char *TheFile);
void ClearStateRecs(void);

/***** TIMING STUFF *******/

#define SECOND_TIME_COUNT 1193000L
#define THIRTY_MINUTES_TIME ((unsigned long)(SECOND_TIME_COUNT*1800))
```

```
#define MILLISECOND_TIME_COUNT 1193L
#define LOOP_DELAY ((unsigned long) (20L * MILLISECOND_TIME_COUNT))
unsigned long longtimer(void);
unsigned DelayUntil(unsigned long TargetTime);
void InitDelayTimer(void);
unsigned long TimeSinceLastDelay(void);
float MS_SinceLastDelay(void);
extern unsigned long LastDelayTime, FirstDelayTime;

#define clearbit(reg,NumBit)  ( (unsigned char)(reg) & (0xFE<<(NumBit)) )
#define setbit(reg,NumBit)  ( (unsigned char)(reg) | (0x01<<(NumBit)) )
#define getbit(reg,NumBit)  ( (unsigned char)(reg) & (0x01<<(NumBit)) )

unsigned int min_b_array[5], max_b_array[5];
/* array to store servo max/min */
void putByte(unsigned char putIt);  /* TXes byte to PC:  Byte TXed is found in
                                       putIt      */
```

Appendix B. MATLAB® .m Files

Listed below are the various MATLAB® .m files used in prototyping the emulator and controller neural networks.

```
function [Wo,Wi,e] = bkprp(x,d,Wo,Wi,eta)

%*************************************************************************
%                  function [Wo,Wi,e] = bkprp(x,d,Wo,Wi,eta)
%
% x -input data vector
% d -desired output vector
% eta - step size
% Wo - output weights
% Wi - input weights
%
%  Backpropagation routine through a NN with a linear input layer, sigmoid
%  transfer function at the hidden layer, and a linear outpur layer.
%*************************************************************************

[p, n1] = size(Wo);  % p - # of output nodes
[n,m] = size(Wi);    % n - # of hidden nodes
                     % m - # of input nodes

%************************Forward propagation******************************

u = Wi*[x ; 1];  %input vector to hidden layer

a = (ones(n,1)+exp(-u)).^(-1);  %output vector of hidden layer

z = Wo*[a ; 1];  %NN output vector

%*************************************************************************

%****************************Backpropagation******************************

e = d-z;   %error at output

deltaEWo = -e*[a ; 1]';   %Equivalent error at hidden layer output

Wo = Wo - eta*deltaEWo;  %Output weight correction

deltaEWi =-([diag(a.*(ones(n,1)-a)), zeros(n,1)])*Wo'*e*[x;1]';
 %Error at input

Wi = Wi - eta*deltaEWi;    %Input weight correction

%************************************************************************
```

```
%*************************************************************************
%                             trntail.m
%
% Iterative routine to train tail rotor emulator.  Calls MATLAB function
% bkprp.m.  Data structure: throttle,rudder,state(k), state(k-1), state(k-2)
%*************************************************************************

clear;
load fame;  % Exemplar file
famenorm =(fame./255);  % Normalize data

Unorm = famenorm(1:300,1:5)';

[m,j] = size(Unorm);   % size of the input vector u - mxj

n = 15;  % # of hidden nodes

D=famenorm(1:300,6)'; % desired output

[p,j]=size(D);

eta=.001; % step size

Wo = rand(p,n+1) -.5;    %creates the input and output weights randomly
Wi = rand(n,m+1) -.5;

l=1;

while l < 10000   %loops through number of epochs

for k = 1:j     %loops through all input vectors

u = Unorm(:,k);

d = D(:,k);

[Wo,Wi,e] = bkprp(u,d,Wo,Wi,eta);

E(:,k) = e;

end

error=sum(.5*(E.^2)');

mserr(l)=error;

l=l+1;

end

save trntail Wo Wi mserr; % save weight and error in file "trntail.mat"
```

```
%*****************************************************************************
%                               check.m
%
%  Forward propagation through NN.  Output is compared with desired and
%  plotted.
%*****************************************************************************

load famenew;
famenorm=famenew./255;


% x -input data vector
% d -desired output vector
% eta - step size
% Wo - output weights
% Wi - input weights

x = famenorm(:,1:5)';

d = famenorm(:,6:8)';

[m,j]=size(x);

[p, n1] = size(Wo);

[n,m] = size(Wi);

% p - # of output nodes
% n - # of hidden nodes
% m - # of input nodes

%************************Forward propagation*****************************

u = Wi*[x ; ones(1,j)];  %input vector to hidden layer

a = (ones(n,j)+exp(-u)).^(-1);  %output vector of hidden layer

z = Wo*[a ; ones(1,j)];  %NN output vector

%*****************************************************************************
plot(1:j,[z;d])

err = .5*sum((z-d).^2)
```

Appendix C. Derivation of Backpropagation

C.1 Emulator

The objective in backpropagation is to minimize error with respect to the neural network weights using a gradient decent technique. In this section, we will first consider backpropagation through the emulator NN with one hidden layer using a sigmoid transfer function

$$f(x) = (1 + e^{-x})^{-1}$$

and linear input and output layers (Figure 21), which will be referred to as a linear-sigmoid-linear NN.

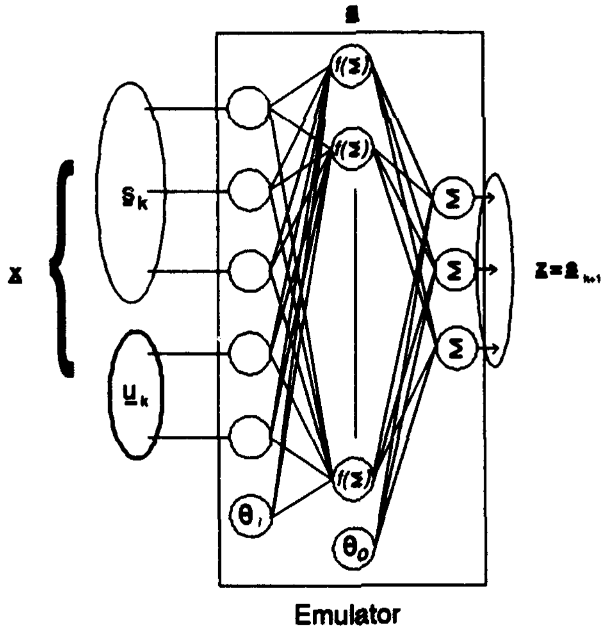


Figure 21. Linear-Sigmoid-Linear NN

This network consists of an input layer with (m x 1) column vector **x**, which is a combination of the control vector **u** and the state vector **s**, and the input bias term θ_i. The output of the hidden layer vector **a** (n x 1), which along with the output bias term θ_o are combined to produce the output vector **z** (p x 1), or the next state of the plant.

In order to simplify the analysis, the input and hidden layer vectors will be combined with the respective bias terms to produce a single vector. Thus the revised input vector is the input vector **x** appended with the input bias term θ_i ($[\mathbf{x}^T|\theta_i]^T$) denoted by $\tilde{\mathbf{x}}$. Similarly, the output of the hidden layer **a** is appended with the output bias term θ_o to produce the revised hidden layer vector $\tilde{\mathbf{a}} = [\mathbf{a}^T|\theta_o]^T$. The associated weight matrices are the input weight matrix $\mathbf{W}_i$ [n x (m+1)] and the output weight matrix $\mathbf{W}_o$ [p x (n+1)].

The derivation of the backpropagation algorithm presented here is a variation of the vanilla backpropagation paradigm used by Rogers and others (16). The analysis is focused on a vector/matrix representation of backpropagation. The output of the network can be represented by the equation:

$$\mathbf{z} = \mathbf{W}_o\tilde{\mathbf{a}} \tag{11}$$

$$\text{where } \tilde{\mathbf{a}} = \left[\frac{\mathbf{a}}{\theta_o}\right] \tag{12}$$

$$\text{and } \mathbf{a} = f_{\mathbf{h}}(\mathbf{W}_i\tilde{\mathbf{x}}) \tag{13}$$

$$\text{and } \tilde{\mathbf{x}} = \left[\frac{\mathbf{x}}{\theta_i}\right] \tag{14}$$

The bias term θ in both the input and output layers is normally set to unity. For the remainder of this derivation, we will consider both θ_i and θ_o to have a value of 1.

The nonlinear vector-valued function $f_{\mathbf{h}}(\cdot)$ in this design shall be referred to as the vector sigmoid function and defined for some arbitrary (n x 1) vector $\underline{\alpha}$ by:

$$f_{\mathbf{h}}(\underline{\alpha}) = [f_h(\alpha_1),\ f_h(\alpha_2),\ \ldots,\ f_h(\alpha_n)]^T \quad (15)$$

$$= [(1+e^{\alpha_1})^{-1},\ (1+e^{\alpha_2}),^{-1}\ \ldots,\ (1+e^{\alpha_n})^{-1}]^T \quad (16)$$

$$(17)$$

The general weight update equation is defined as:

$$\mathbf{W}^+ = \mathbf{W}^- - \eta\frac{\partial E}{\partial \mathbf{W}^-} \quad (18)$$

$$\text{where } E = \frac{1}{2}||\mathbf{d} - \mathbf{z}||_2^2 \quad (19)$$

where the desired output of the NN is represented by the vector **d** and the output of the net is represented by **z**.

C.1.1 Output Layer. Consider the problem of minimizing the error function $\frac{1}{2}||\mathbf{d} - \mathbf{z}||_2^2$ with respect to the input output weight matrix $\mathbf{W}_o$. Begin by taking the partial derivative of the error function with respect to the output weight matrix $\mathbf{W}_o$.

$$\frac{\partial E}{\partial \mathbf{W}_o} = \frac{1}{2}\frac{\partial}{\partial \mathbf{W}_o}||\mathbf{d} - \mathbf{z}||_2^2 \quad (20)$$

$$= \frac{1}{2}\frac{\partial}{\partial \mathbf{W}_o}(\mathbf{d} - \mathbf{z})^T(\mathbf{d} - \mathbf{z}) \quad (21)$$

$$= \frac{1}{2}\frac{\partial}{\partial \mathbf{W}_o}(\mathbf{d}^T\mathbf{d} - 2\mathbf{z}^T\mathbf{d} + \mathbf{z}^T\mathbf{z}) \quad (22)$$

$$= \frac{1}{2}\frac{\partial}{\partial \mathbf{W}_o}(\mathbf{d}^T\mathbf{d} - 2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d} + \tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\mathbf{a}) \quad (23)$$

$$= \frac{1}{2}\left[\frac{\partial}{\partial \mathbf{W}_o}(\mathbf{d}^T\mathbf{d}) + \frac{\partial}{\partial \mathbf{W}_o}(-2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) + \frac{\partial}{\partial \mathbf{W}_o}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\tilde{\mathbf{a}})\right] \quad (24)$$

Since $\mathbf{d}$ is not a function of $\mathbf{W}_o$, then $\frac{\partial}{\partial \mathbf{W}_o}(\mathbf{d}^T\mathbf{d}) = 0$. Thus, there are only two remaining terms to consider. In order to more easily analyze the partial derivatives, it might be helpful to restructure the matrix into a row vector with column vectors as elements.

$$\frac{\partial}{\partial \mathbf{W}_o}(\underline{\alpha}) = \left[\frac{\partial}{\partial \mathbf{w}_{o_1}}(\underline{\alpha}), \frac{\partial}{\partial \mathbf{w}_{o_2}}(\underline{\alpha}), \ldots, \frac{\partial}{\partial \mathbf{w}_{o_n}}(\underline{\alpha})\right] \tag{25}$$

Examining a single column vector gradient $\frac{\partial}{\partial \mathbf{w}_{o_k}}(\underline{\alpha})$, where $k \in \{1, \cdots, n, n+1\}$, consider the last term in equation 24:

$$\frac{\partial}{\partial \mathbf{w}_{o_k}}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\tilde{\mathbf{a}}) = \frac{\partial}{\partial \mathbf{w}_{o_k}}\left[(\mathbf{W}_o\tilde{\mathbf{a}})^T\mathbf{W}_o\tilde{\mathbf{a}}\right] \tag{26}$$

Applying the product rule for a vector gradient (17:274), where $\mathbf{a}$ and $\mathbf{b}$ are vector-valued functions of the vector $\underline{\theta}$:

$$\frac{\partial}{\partial \underline{\theta}}\tilde{\mathbf{a}}^T(\underline{\theta})\,\mathbf{b}(\underline{\theta}) = \left(\frac{\partial}{\partial \underline{\theta}}\mathbf{a}^T(\underline{\theta})\right)\mathbf{b}(\underline{\theta}) + \left(\frac{\partial}{\partial \underline{\theta}}\mathbf{b}^T(\underline{\theta})\right)\tilde{\mathbf{a}}(\underline{\theta}) \tag{27}$$

If $\mathbf{a}(\underline{\theta}) = \mathbf{b}(\underline{\theta})$, then the equation becomes:

$$\frac{\partial}{\partial \underline{\theta}}\tilde{\mathbf{a}}^T(\theta)\,\mathbf{a}(\underline{\theta}) = 2\left(\frac{\partial}{\partial \theta}\mathbf{a}^T(\underline{\theta})\right)\mathbf{a}(\underline{\theta}) \tag{28}$$

We can then express equation 26 as:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}_{o_k}}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\tilde{\mathbf{a}}) &= 2\left(\frac{\partial}{\partial \mathbf{w}_{o_k}}(\mathbf{W}_o\tilde{\mathbf{a}})^T\right)\mathbf{W}_o\tilde{\mathbf{a}} && (29)\\ &= 2\left(\frac{\partial}{\partial \mathbf{w}_{o_k}}[a_1\mathbf{w}_{o_1}^T + a_2\mathbf{w}_{o_2}^T + \cdots + a_n\mathbf{w}_{o_n}^T + \mathbf{w}_{o_{n+1}}^T]\right)\mathbf{W}_o\tilde{\mathbf{a}} && (30)\end{aligned}$$

Then taking the gradient, we find only a_k remains, and thus equation 26 becomes:

$$\frac{\partial}{\partial \mathbf{w}_{o_k}}(\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{W}_o \tilde{\mathbf{a}}) = \begin{cases} 2a_k \mathbf{W}_o \tilde{\mathbf{a}} & 1 \leq k \leq n \\ 2\mathbf{W}_o \tilde{\mathbf{a}} & k = n+1 \end{cases} \tag{31}$$

And then reconstructing the matrix:

$$\frac{\partial}{\partial \mathbf{W}_o}(\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{W}_o \tilde{\mathbf{a}}) = [2a_1 \mathbf{W}_o \tilde{\mathbf{a}} \;\; 2a_2 \mathbf{W}_o \tilde{\mathbf{a}} \cdots 2a_n \mathbf{W}_o \tilde{\mathbf{a}}] \tag{32}$$

$$= 2\mathbf{W}_o \tilde{\mathbf{a}} \tilde{\mathbf{a}}^T \tag{33}$$

Examining the second term of the equation 24:

$$\frac{\partial}{\partial \mathbf{W}_o}(-2\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{d}) = \left[\frac{\partial(-2\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{d})}{\partial \mathbf{w}_{o_1}} \; \frac{\partial(-2\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{d})}{\partial \mathbf{w}_{o_2}} \cdots \frac{\partial(-2\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{d})}{\partial \mathbf{w}_{o_n}} \right] \tag{34}$$

Again, examining a single vector gradient for the kth vector:

$$\frac{\partial}{\partial \mathbf{w}_{o_k}}(-2\tilde{\mathbf{a}}^T \mathbf{W}_o^T \mathbf{d}) = \frac{\partial}{\partial \mathbf{w}_{o_k}}[a_1 \; a_2 \cdots a_n \; 1][\mathbf{w}_{o_1}^T \; \mathbf{w}_{o_2}^T \cdots \mathbf{w}_{o_n}^T \; \mathbf{w}_{o_{n+1}}^T]^T \mathbf{d} \tag{35}$$

$$= \frac{\partial}{\partial \mathbf{w}_{o_k}}[a_1 \mathbf{w}_{o_1}^T + a_2 \mathbf{w}_{o_2}^T + \cdots + a_n \mathbf{w}_{o_n}^T + \mathbf{w}_{o_{n+1}}^T]\mathbf{d} \tag{36}$$

$$= \begin{cases} a_k \mathbf{d} & 1 \leq k \leq n \\ \mathbf{d} & k = n+1 \end{cases} \tag{37}$$

Placing the vectors back into matrix form:

$$\frac{\partial}{\partial \mathbf{W}_o}(-2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) = -2[a_1\mathbf{d}\ a_2\mathbf{d}\cdots a_n\mathbf{d}\ \mathbf{d}] \tag{38}$$

$$= -2\mathbf{d}\tilde{\mathbf{a}}^T \tag{39}$$

Combining the two portions of the original equation 24:

$$\frac{\partial E}{\partial \mathbf{W}_o} = \frac{1}{2}\left[\frac{\partial}{\partial \mathbf{W}_o}(-2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) + \frac{\partial}{\partial \mathbf{W}_o}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\mathbf{a})\right] \tag{40}$$

$$= -\mathbf{d}\tilde{\mathbf{a}}^T + \mathbf{W}_o\tilde{\mathbf{a}}\tilde{\mathbf{a}}^T \tag{41}$$

$$= -(\mathbf{d} - \mathbf{W}_o\tilde{\mathbf{a}})\tilde{\mathbf{a}}^T \tag{42}$$

$$= -(\mathbf{d} - \mathbf{z})\tilde{\mathbf{a}}^T \tag{43}$$

C.1.2 Input Layer. Now to address correction of the input network weights, $\mathbf{W}_i$. We begin as before by taking the gradient of the error function $E = \frac{1}{2}||\mathbf{d} - \mathbf{z}||_2^2$ with respect to $\mathbf{W}_i$. In the same way as the output weight derivative, we arrive at the following:

$$\frac{\partial E}{\partial \mathbf{W}_i} = \frac{\partial}{\partial \mathbf{W}_i}\frac{1}{2}||\mathbf{d} - \mathbf{z}||_2^2 \tag{44}$$

$$= \frac{1}{2}\left[\frac{\partial}{\partial \mathbf{W}_i}(\mathbf{d}^T\mathbf{d}) + \frac{\partial}{\partial \mathbf{W}_i}(-2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) + \frac{\partial}{\partial \mathbf{W}_i}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\tilde{\mathbf{a}})\right] \tag{45}$$

And as before, we examine the kth column of the gradient:

$$\frac{\partial}{\partial \mathbf{W}_i}(\underline{\alpha}) = \left[\frac{\partial}{\partial \mathbf{w}_{i_1}}(\underline{\alpha})\ \frac{\partial}{\partial \mathbf{w}_{i_2}}(\underline{\alpha})\cdots\frac{\partial}{\partial \mathbf{w}_{i_{m+1}}}(\underline{\alpha})\right] \tag{46}$$

Using definition $\frac{\partial}{\partial \underline{\theta}}\mathbf{m}^T\mathbf{Q}\mathbf{m} = 2(\frac{\partial}{\partial \underline{\theta}}\mathbf{m}^T)\mathbf{Q}\mathbf{m}$ (17:274), the last term in equation 45 becomes

$$\frac{\partial}{\partial \mathbf{w}_{i_h}}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{W}_o\tilde{\mathbf{a}}) = 2\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)\mathbf{W}_o^T\mathbf{W}_o f_{\mathbf{h}}(\mathbf{W}_i\tilde{\mathbf{x}}) \tag{47}$$

Turning attention to the second term in equation 45 in a similar vector gradient analysis:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}_{i_h}}(-2\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) &= -2\frac{\partial}{\partial \mathbf{w}_{i_h}}(\tilde{\mathbf{a}}^T\mathbf{W}_o^T\mathbf{d}) && (48)\\ &= -2\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)\mathbf{W}_o^T\mathbf{d} && (49)\\ & && (50)\end{aligned}$$

Notice both terms of equation 45 have the common factor $2(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T)\mathbf{W}_o^T$. Recombining the two terms and factoring:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{w}_{i_h}} &= -2\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)\mathbf{W}_o^T(\mathbf{d}-\mathbf{W}_o\tilde{\mathbf{a}}) && (51)\\ &= -2\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)\mathbf{W}_o^T(\mathbf{d}-\mathbf{W}_o\tilde{\mathbf{a}}) && (52)\\ &= -2\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z}) && (53)\end{aligned}$$

Focusing attention on the gradient calculation $\left(\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T\right)$ we now analyze the gradient of the vector sigmoid function. For simplicity, let $\underline{\alpha} = \mathbf{W}_i\tilde{\mathbf{x}}$.

$$\frac{\partial}{\partial \mathbf{w}_{i_h}}\tilde{\mathbf{a}}^T = \frac{\partial}{\partial \mathbf{w}_{i_h}}[f_{\mathbf{h}}^T(\mathbf{W}_i\tilde{\mathbf{x}})|1] \tag{54}$$

$$= \frac{\partial}{\partial \mathbf{w}_{i_k}} [f_{\mathbf{h}}^T(\alpha)|1] \tag{55}$$

$$= \frac{\partial}{\partial \mathbf{w}_{i_k}} [f_h(\alpha_1),\ f_h(\alpha_2),\ldots,f_h(\alpha_n)|1] \tag{56}$$

$$= \frac{\partial}{\partial \mathbf{w}_{i_k}} [f_h(\alpha_1),\ f_h(\alpha_2),\ldots,f_h(\alpha_n)]|\frac{\partial}{\partial \mathbf{w}_{i_k}}(1) \tag{57}$$

$$= \frac{\partial}{\partial \mathbf{w}_{i_k}} [f_h(\alpha_1),\ f_h(\alpha_2),\ldots,f_h(\alpha_n)]|0 \tag{58}$$

$$= \begin{bmatrix} \frac{\partial f_h(\alpha_1)}{\partial w_{i_{1k}}} & \frac{\partial f_h(\alpha_2)}{\partial w_{i_{1k}}} & \cdots & \frac{\partial f_h(\alpha_n)}{\partial w_{i_{1k}}} & 0 \\ \frac{\partial f_h(\alpha_1)}{\partial w_{i_{2k}}} & \frac{\partial f_h(\alpha_2)}{\partial w_{i_{2k}}} & \cdots & \frac{\partial f_h(\alpha_n)}{\partial w_{i_{2k}}} & 0 \\ & & \vdots & & \\ \frac{\partial f_h(\alpha_1)}{\partial w_{i_{nk}}} & \frac{\partial f_h(\alpha_2)}{\partial w_{i_{nk}}} & \cdots & \frac{\partial f_h(\alpha_n)}{\partial w_{i_{nk}}} & 0 \end{bmatrix} \tag{59}$$

Examining each element,

$$\frac{\partial f_h(\alpha_\ell)}{\partial w_{i_{jk}}} = \frac{\partial}{\partial w_{i_{jk}}} (1+e^{-\alpha_\ell})^{-1}) \tag{60}$$

$$= -(1+e^{-\alpha_\ell})^{-2} e^{-\alpha_\ell}(-1)\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} \tag{61}$$

$$= (1+e^{-\alpha_\ell})^{-1} e^{-\alpha_\ell}(1+e^{-\alpha_\ell})^{-1}\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} \tag{62}$$

$$= f_h(\alpha_\ell)(1-f_h(\alpha_\ell))\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} \tag{63}$$

$$= a_\ell(1-a_\ell)\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} \tag{64}$$

we find that

$$\frac{\partial}{\partial \mathbf{w}_{i_k}} \tilde{\mathbf{a}} = \begin{bmatrix} a_1(1-a_1)\frac{\partial \alpha_1}{\partial w_{i_{1k}}} & a_2(1-a_2)\frac{\partial \alpha_2}{\partial w_{i_{1k}}} \cdots a_n(1-a_n)\frac{\partial \alpha_n}{\partial w_{i_{1k}}} & 0 \\ a_1(1-a_1)\frac{\partial \alpha_1}{\partial w_{i_{2k}}} & a_2(1-a_2)\frac{\partial \alpha_2}{\partial w_{i_{2k}}} \cdots a_n(1-a_n)\frac{\partial \alpha_n}{\partial w_{i_{2k}}} & 0 \\ & \vdots & \\ a_1(1-a_1)\frac{\partial \alpha_1}{\partial w_{i_{nk}}} & a_2(1-a_2)\frac{\partial \alpha_2}{\partial w_{i_{nk}}} \cdots a_n(1-a_n)\frac{\partial \alpha_n}{\partial w_{i_{nk}}} & 0 \end{bmatrix} \tag{65}$$

However, $\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} = 0$ when $\ell \neq k$. This can be seen when considering the the vector $\underline{\alpha}$:

$$\begin{aligned} \underline{\alpha} &= \mathbf{W}_i \tilde{\mathbf{x}} && (66) \\ &= [\mathbf{w}_{i_1} x_1 + \mathbf{w}_{i_2} x_2 + \cdots + \mathbf{w}_{i_m} x_m + \mathbf{w}_{i_{m+1}}] && (67) \end{aligned}$$

The ℓth element of this vector is:

$$\alpha_\ell = w_{i_{\ell 1}} x_1 + w_{i_{\ell 2}} x_2 + \cdots + w_{i_{\ell m}} x_m + w_{i_{\ell(m+1)}} \qquad (68)$$

When taking the derivative with respect to $w_{i_{jk}}$, the only term in the sum that remains is the kth term when $\ell = j$. Thus:

$$\frac{\partial \alpha_\ell}{\partial w_{i_{jk}}} = \begin{cases} x_k & \ell = j \\ 0 & \ell \neq j \end{cases} \qquad (69)$$

The matrix in equation 65 then reduces to a diagonal matrix

$$\frac{\partial}{\partial \mathbf{w}_{i_k}} \tilde{\mathbf{a}} = \begin{bmatrix} a_1(1-a_1)x_k & 0 & \cdots & 0 & 0 \\ 0 & a_2(1-a_2)x_k & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_n(1-a_n)x_k & 0 \end{bmatrix} \qquad (70)$$

$$(71)$$

$$\begin{aligned} &= [diag[a_1(1-a_1)x_k,\ a_2(1-a_2)x_k,\ \cdots,\ a_n(1-a_n)x_k] \mid \underline{0}_n] && (72) \\ &= [diag\,(\mathbf{a} \odot (\underline{1}_n - \mathbf{a}))\ \mid \underline{0}_n)]\, x_k && (73) \end{aligned}$$

where $\odot$ represent the Hadamard or array product (10:45). Inserting this result into equation 53:

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{w}_{i_k}} &= -2\left(\frac{\partial}{\partial \mathbf{w}_{i_k}}\tilde{\mathbf{a}}\right)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z}) && (74)\\ &= -diag\left(\mathbf{a}\odot(\underline{1}_n-\mathbf{a})\right) \mid \underline{0}_n)x_k\mathbf{W}_o^T(\mathbf{d}-\mathbf{z}) && (75)\\ &= -diag\left(\mathbf{a}\odot(\underline{1}_n-\mathbf{a})\right) \mid \underline{0}_n)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})x_k && (76)\end{aligned}$$

Combining all the vectors into the original matrix in equation 45:

$$\frac{\partial E}{\partial \mathbf{W}_i} = \begin{bmatrix} -(diag[\mathbf{a}\odot(\underline{1}_n-\mathbf{a})] \mid \underline{0}_n)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})x_1 \\ -(diag[\mathbf{a}\odot(\underline{1}_n-\mathbf{a})] \mid \underline{0}_n)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})x_2 \\ \vdots \\ -(diag[\mathbf{a}\odot(\underline{1}_n-\mathbf{a})] \mid \underline{0}_n)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})x_m \\ -(diag[\mathbf{a}\odot(\underline{1}_n-\mathbf{a})] \mid \underline{0}_n)\mathbf{W}_o^T(\mathbf{d}-\mathbf{z}) \end{bmatrix}^T \quad (77)$$

$$(78)$$

$$\begin{aligned} &= -\left[diag\left(\mathbf{a}\odot(\underline{1}_n-\mathbf{a})\right)|\underline{0}_n\right]\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})[x_1\ x_2\ \cdots\ x_m\ 1] && (79)\\ &= -\left[diag\left(\mathbf{a}\odot(\underline{1}_n-\mathbf{a})\right)|\underline{0}_n\right]\mathbf{W}_o^T(\mathbf{d}-\mathbf{z})\tilde{\mathbf{x}}^T && (80)\end{aligned}$$

C.1.3 Summary. The equation for forward propagation through the linear-sigmoid-linear neural network is:

$$\begin{aligned}\mathbf{z} &= \mathbf{W}_o\tilde{\mathbf{a}} && (81)\\ &\text{where } \tilde{\mathbf{a}} = \begin{bmatrix}\mathbf{a}\\ \hline \mathbf{1}\end{bmatrix} && (82)\\ &\text{and } \mathbf{a} = f_{\mathrm{h}}(\mathbf{W}_i\tilde{\mathbf{x}}) && (83)\\ &\text{and } \tilde{\mathbf{x}} = \begin{bmatrix}\mathbf{x}\\ \hline \mathbf{1}\end{bmatrix} && (84)\end{aligned}$$

And the backpropagation weight update equations are:

$$\mathbf{W}_o^+ = \mathbf{W}_o^- - \eta\frac{\partial E}{\partial \mathbf{W}_o^-} \tag{85}$$

$$\text{where } \frac{\partial E}{\partial \mathbf{W}_o^-} = -(\mathbf{d}-\mathbf{z})\tilde{\mathbf{a}}^T \tag{86}$$

$$\mathbf{W}_i^+ = \mathbf{W}_i^- - \eta\frac{\partial E}{\partial \mathbf{W}_i^-} \tag{87}$$

$$\text{where } \frac{\partial E}{\partial \mathbf{W}_i^-} = -[diag(\mathbf{a}\odot(\underline{1}_n-\mathbf{a}))|\underline{0}_n]\mathbf{W}_o^{-T}(\mathbf{d}-\mathbf{z})\tilde{\mathbf{x}}^T \tag{88}$$

C.2 Controller

The controller network in this design is also a linear-sigmoid-linear neural network. The output of this network is the control vector **u**. This is combined with the state vector **s** to produce the input for the emulator (Figure 22).

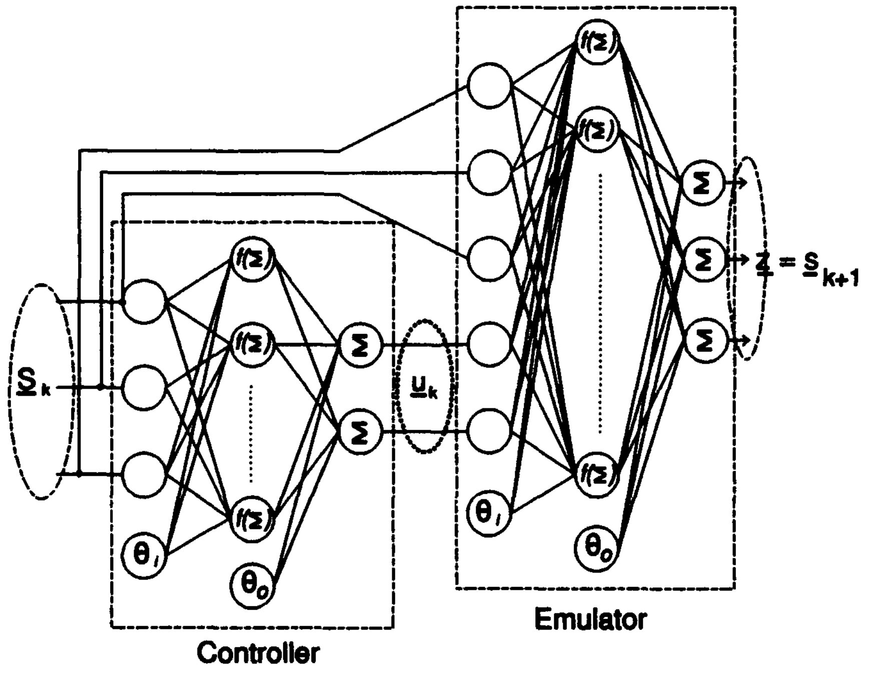


Figure 22. Emulator and Controller

C.2.1 Equivalent Error. The backpropagation derivation for the controller network is identical to the emulator, with the exception of the origin of the error. In the case of the emulator, there is a specific desired response to be compared to the output of the net. In the controller, however, there is no direct measure of error. In this instance, the concept of equivalent error is used. Examining a slightly different representation of the weight update equation used by Nguyen (13):

$$\mathbf{W}^{+} = \mathbf{W}^{-} + 2\mu\underline{\delta}\mathbf{v}^{T} \tag{89}$$

where $\underline{\delta}$ is the equivalent error vector defined by

$$\underline{\delta}_{o} = (\mathbf{d}-\mathbf{z})f'_{\mathbf{o}}(\mathbf{W}_{o}\mathbf{a}) \tag{90}$$

$$\underline{\delta}_{h} = f'_{\mathbf{h}}(\mathbf{W}_{o}\mathbf{a})\mathbf{W}_{o}^{T}\underline{\delta}_{o} \tag{91}$$

for the output and hidden layers respectively and $\mathbf{v}$ is the output of the previous layer. Thus we can alternatively represent the weight update equations derived in section C.1 as:

$$\mathbf{W}_{o}^{+} = \mathbf{W}_{o}^{-} + \eta\underline{\delta}_{o}\tilde{\mathbf{a}}^{T} \tag{92}$$

$$\mathbf{W}_{i}^{+} = \mathbf{W}_{i}^{-} + \eta\underline{\delta}_{h}\tilde{\mathbf{x}}^{T} \tag{93}$$

Again, $\underline{\delta}_{o}$ and $\underline{\delta}_{h}$ represent the equivalent errors at the output and hidden nodes respectively This is identical to the derivation in section 1. Note that the equivalent error at the output layer of the emulator is:

$$\begin{aligned} \underline{\delta}_{o_e} &= (\mathbf{d} - \mathbf{s}) && (94) \\ & \quad \text{since } f'_{\mathbf{o}}(\mathbf{W}_o\mathbf{a}) = 1 && (95) \end{aligned}$$

and the equivalent error at the hidden layer is:

$$\begin{aligned} \underline{\delta}_{h_e} &= f'_{\mathbf{h}}(\mathbf{W}_i\mathbf{x})\mathbf{W}_o^T\underline{\delta}_{o_e} && (96) \\ &= (diag(\mathbf{a} \odot (\underline{1}_n - \mathbf{a})|\underline{0}_n)\mathbf{W}_o^{-T}(\mathbf{d} - \mathbf{z}) && (97) \\ & \quad \text{where } f'_{\mathbf{h}}(\mathbf{W}_i\mathbf{x}) = [diag(\mathbf{a} \odot (\underline{1}_n - \mathbf{a}))|\underline{0}_n] && (98) \end{aligned}$$

C.2.2 Controller Weights. In order to update the controller weights, the final state error at the emulator output is backpropagated through the emulator net without updating the emulator weights. At the output to each layer, there is an equivalent error which is used by the preceding layer as the error.

In the output layer of the emulator net, the equivalent error is simply the actual error. The hidden lay prior to this, the error is distributed over each output of the hidden layer. Since the input to the emulator NN, **x**, consists of both the state vector **s** and the control vector **u**, the equivalent error will also consists of two components, $\underline{\delta}_s$ and $\underline{\delta}_u$. Thus, the equivalent error at the output of the controller net would be the equivalent error for the control vector **u** at the input to the emulator net.

$$\begin{aligned} \underline{\delta}_{o_c} &= \underline{\delta}_u && (99) \\ &= f'_{\mathbf{o}_e}(\mathbf{W}_{o_c})\left\{\mathbf{W}_{i_e}^T\underline{\delta}_{h_e}\right\}_u && (100) \end{aligned}$$

The subscript u in Equation 100 indicates those elements of the vector corresponding to the control vector $\mathbf{u}$. Applying the same approach used in the emulator NN, the equivalent error at the hidden layer of the controller NN is:

$$\begin{aligned} \underline{\delta}_{h_c} &= f_{\mathbf{h}}'(\mathbf{W}_{i_c}\mathbf{s})\mathbf{W}_{o_c}^T\delta_{o_c} && (101) \\ &= (diag(\mathbf{a}_c \odot (\underline{1}_n - \mathbf{a}_c)|\underline{0}_n)W_{o_c}^T\delta_{o_c} && (102) \end{aligned}$$

C.2.3 Summary. The controller network update equations are:

$$\begin{aligned} \mathbf{W}_{o_c}^+ &= \mathbf{W}_{o_c}^- - \eta\underline{\delta}_{o_c}\tilde{\mathbf{a}}_c && (103) \\ & \underline{\delta}_{o_c} = \{\mathbf{W}_{i_e}^T\underline{\delta}_{h_e}\}_u && (104) \\ & \underline{\delta}_{h_e} = (diag(\mathbf{a}_e \odot (\underline{1}_n - \mathbf{a}_e)|\underline{0}_n)\mathbf{W}_{o_e}^{-T}\underline{\delta}_{i_e} && (105) \\ & \underline{\delta}_{i_e} = (\mathbf{d} - \mathbf{z}) && (106) \\ \mathbf{W}_{i_c}^+ &= \mathbf{W}_{i_c}^- - \eta\underline{\delta}_{h_c}\tilde{\mathbf{s}} && (107) \\ & \underline{\delta}_{h_c} = (diag(\mathbf{a}_c \odot (\underline{1}_n - \mathbf{a}_c)|\underline{0}_n)\mathbf{W}_{o_c}^{-T}\underline{\delta}_{o_c} && (108) \end{aligned}$$

C.3 Back Propagation Through Time

In the case of BPTT, this process of equivalent error can continue throughout the previous time steps with a slight modification. Recall in obtaining Equation 100 there also is a component of the equivalent error corresponding to the state input vector $\mathbf{z}$. This is the same vector that is simultaneously input to the controller NN.

The emulator output of the previous state is input to both the emulator and control NNs of the next state. Each output node of the emulator is connected to an input node of both the

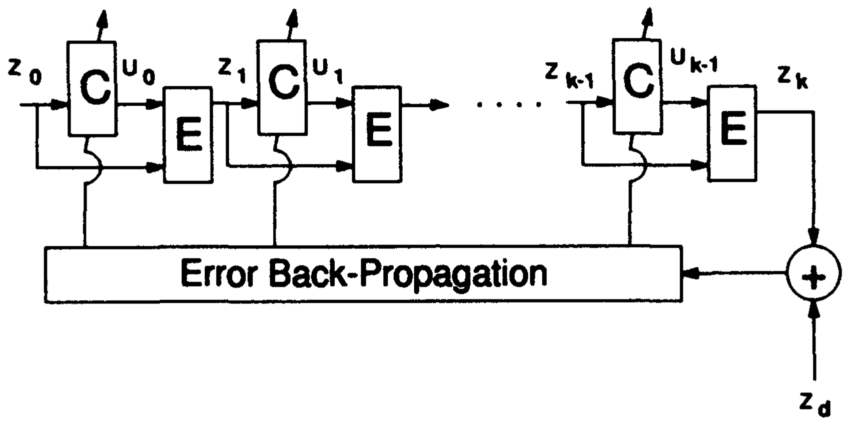


Figure 23. Training the Controller (C = Controller, E = Emulator) (13:20)

emulator and controller of the next state. If we examine the backpropagation equations as applied to a single node, we find the equivalent error is the sum of the equivalent errors

$$\begin{aligned} \delta_{j(k-1)} &= f'(x)\mathbf{w}_j^T \underline{\delta}_{(k)} && (109) \\ &= f'(x)\sum_{i=1}^{n} w_{ij}\delta_{i(k)} && (110) \end{aligned}$$

Since there are no weights or nonlinearities between states, this reduces to:

$$\delta_{j(k-1)} = \sum_{i=1}^{n} \delta_{i(k)} \qquad (111)$$

Thus, the error vector at the output of the emulator in the $(k-1)$th state $(\underline{\delta}_{o_{e(k-1)}})$ is the sum of the equivalent errors for the position vector $(\mathbf{s})$ at the input of the emulator $(\underline{\delta}_{i_{e_s(k)}})$ and controller $(\underline{\delta}_{i_{c(k)}})$ in the kth state.

$$\underline{\delta}_{o_{e(k-1)}} = \underline{\delta}_{i_{e_s(k)}} + \underline{\delta}_{i_{c(k)}} \qquad (112)$$

Then it simply becomes a matter of backpropagating this error through the $(k-1)$th state and continuing the process until reaching the first state.

Appendix D. FAME

This appendix includes documentation on the various hardware and software modifications made in the course of this research, changes to the Report on the Fast Adaptive Maneuvering Experiment (FAME) (Section D.3.1), and the user's guide for the AFIT FAME apparatus (Section D.4). Prior to attempting any experimentation using FAME, it is highly recommended the experimenter read the entire contents of the original report and as well as all updates. Current software code as well as various documents are available through anonymous FTP to "fame.gmu.edu" under the directory "pub". Section D.2 includes a reprint of the various FAME documents available through FTP. The primary point of contact at George Mason University is Darrell Duane, E-Mail dduane@mason1.gmu.edu.

D.1 Summary of Updates

D.1.1 Software. The FAME apparatus was delivered with the software FAMEPC, version 3.0. Since then, versions 4.0 through 6.0 have been made available. Also available is a C++ Windows version of the software called TEMPO1. A version of TEMPO1 for MS-DOS is also in production. In this research, FAMEPC, version 6.0 was used with some minor modifications. A listing of this software can be found in Appendix A

Minor changes to version 6.0 of the software included changing the directory paths of the header files "pcdef.h" and "famedef.h". A small section of code was changed to select of COM 1 instead of COM 2 as the communications port interface to the MCU. Finally, the menu display was changed to reflect the correct software version and to correct a spelling error.

A change was also made in the timing function "DelayUntil()" in "ControlLoop().". This function caused intermittent program lockup while attempting manually control the FAME helicopter. This function was replaced with the C-standard function "delay(10)", which produced a time delay of 10 milliseconds. Unfortunately, this causes inaccurate timing data to be recorded in "file1.trn." However, the data points are still recorded at even intervals of approximately 25 milliseconds.

It is not mentioned in any of the FAME documentation, but it is also necessary to include a file called "generic.cal" in the same directory as FAMEPC.EXE. The program will run without it, but it will not report the state variables.

D.1.2 Hardware. A number of changes to the FAME hardware were made in the course of this research. Some were at the direction of the FAME developers at GMU while others were created out of necessity specific to this research.

D.1.3 Directed Changes. Changes directed by the FAME developers are fully documented in Section D.2.

D.1.3.1 Local Changes. An additional hardware modification was necessary to correct a defect in the apparatus. The aileron servo was not operational du、 to an open circuit in one of the leads. Since it turned out this lead was common to all the servos, the correction was to install a jumper from an adjacent connector. The jumper is between connectors 1 and 2 on the servo block.

Quick connects were also installed to allow easy removal of the helicopter from the test stand. The connectors are standard for Radio Controlled use and can be easily connected to a standard helicopter radio receiver.

The Kalt Whisper helicopter is secured to the stand using plastic tie wraps. Again, this permits easy removal of the helicopter by simply cutting the ties wraps. These tie wraps are inexpensive an easily replaced.

It was also necessary to modify the stand to stabilize the helicopter during take off and landing. As delivered, the helicopter remained suspended from the training platform during the take-off run-up. The engine torque would cause the helicopter to yaw wildly until the tail rotor developed enough speed to generate yaw control. Similar problems resulted on landing. The modification raises the red direction of flight (DOF) collar to support the landing skids when the apparatus is resting on the floor. As the helicopter begins to rise, the platform lowers, allowing freedom of movement about all axes. Drawings of this modification can be found in Section D.3.1.

D.2 Fast Adaptive Maneuvering Experiment File Repository

The following section is a reprint of the various FAME documents available by anonymous FTP to fame.gmu.edu. Footnoted comments are provided to document the changes in the AFIT FAME apparatus and to help further clarify the updates.

D.2.1 GYROMOD.W51: Hardware Modification to facilitate pulse width measurement from Gyro. In order to enable the software to measure the length of the pulse width from the Gyro which provides an indication of the rate of rotation of the helicopter (Yaw Dot), a wire must be replaced on the servo connector block. The quick ties which secure the bulk of the ribbon cable to the aluminum extension should be clipped. The ribbon cable follows the resistor color code for the units digit of the pin number, thus pin #28 is a gray wire. It is this wire which should be unsoldered from servo block position #0, and the remaining uninsulated wire should be clipped.

The new wire that is to be connected at the servo block position #0 is pin #32, a red wire, and can be found by looking 4 wires above the unsoldered gray wire. It is not necessary to place the new wire into the sheath where the remaining servo block wires are protected. However, a piece of heat shrink tubing is provided to insulate at the solder joint. Also, quick ties are provided to secure the ribbon cable to the aluminum extension. This completes the hardware modification to gather data from the Gyro.[1]

[1]These changes are already installed on the AFIT FAME apparatus.

D.2.2 README: Fast Adaptive Maneuvering Experiment File Repository . The files for all previous version of the FAME project are available individually or as a set in various files fame5_0.tar.Z in the directory ver5_0.

To access these files, FTP them in the binary mode (type binary at the FTP prompt), use the uncompress command to expand the file, (type uncompress fame5_0.tar.Z), untar the files (tar xfv fame5_0.tar), and the files will be available on your unix machine.[2]

Also, the FAME report is available in postscript format in the report directory.

D.2.3 WJREAD.TXT. File name: WJREAD.TXT

Created: 3/5/92

Updated: 3/26/92

This text file gives instructions on how to upload WRITEJMP.0 to the Motorola MC68HC11EVB. It is assumed that the user is familiar with Kermit and/or PROCOMM or some other terminal emulation program.

WRITEJMP.0 programs the first three memory locations in EEPROM to jump to the start of FAMEMAIN. This jump will occur if jumper J4 on the EVB is connected between pins 2-3. If J4 is connected between pins 1-2, then the Buffalo monitor will start when the restart button is pushed.

If WRITEJMP.0 has been run on the EVB, then jumper J4 on the board must be moved to the 1-2 position so that the Buffalo monitor will run, and then switched back to 2-3 after FAMEMAIN.0 is uploaded.

Assuming Kermit:

1. Connect the PC without a null modem to the bottom DB-25 connector on the EVB. 9600 Baud, No parity, 1 Stop bit.

[2]The same procedure applies to any of the tarred files available from the FAME repository.

2. Start Kermit. Connect. If no Buffalo prompt (>), push reset button on EVB (located below tne 4 screw contact power connector).

3. At the ">" prompt, type `load t` followed by a carriage return.

4. Escape back to Kermit by pressing the keys `control-]` simultaneously followed by the single letter c

5. At the kermit prompt, type `transmit WRITEJMP.O \O`. The `\O` is important as it disables some handshaking.

6. It takes several minutes to download.

7. When the Kermit prompt appears, type `c` to connect to the EVB again.

8. Type the keys `control-A` simultaneously followed by a carriage return. The ">" prompt should reappear.

9. One must now either type `g c000` to cause the WRITEJMP program to run. It only takes a few seconds.

10. If there is no prompt after running WRITEJMP, push the restart button and determine whether the automatic startup is enabled by typing, at the Buffalo prompt, >, `md b600` followed by a carriage return. Buffalo should respond with several lines containing the contents of the memory starting at location 0xB600. The first line should contain `7E 18 00`. This can only be programmed by running the writejmp.0 program since it is in EEPROM.

The procedure in PROCOMM is similar except the file is uploaded using the page-up key, ASCII transfer (menu number 7), and then typing the filename. When finished, the Buffalo prompt will appear and there is no need to type control-A. On some machines, PROCOMM is terribly slow in uploading and Kermit is much faster.[3]

[3]A reminder that all communication occurs at 19200 baud using the lower DB-25 connector. At this speed, the procedure using PROCOMM and ASCII transfer proved to be quite effective, especially when using a 386-based machine. Transfers normally took about 30-60 seconds.

D.2.4 FMREAD.TXT. George Mason University, Department of Electrical and Computer Engineering, 4400 University Drive, Fairfax, VA 22030

File name: FMREAD.TXT

Created: 3/5/92

Updated: 3/26/92

This text file gives instructions on how to upload FAMEMAIN.0 to the Motorola MC68HC11EVB. It is assumed that the user is familiar with Kermit and/or PROCOMM or some other terminal emulation program.

If WRITEJMP.0 has been run on the EVB, then jumper J4 on the board must be moved to the 1-2 position so that the Buffalo monitor will run, and then switched back to 2-3 after FAMEMAIN.0 is uploaded.

Assuming Kermit:

1. Connect the PC without a null modem to the bottom DB-25 connector on the EVB. 9600 Baud, No parity, 1 Stop bit.
2. Start Kermit. Connect. If no Buffalo prompt (>), push reset button on EVB (located below the 4 screw contact power connector).
3. At the ">" prompt, type `load t` followed by a carriage return.
4. Escape back to Kermit by pressing the keys `control-]` simultaneously followed by the single letter [illegible]
5. At the kermit prompt, type `transmit FAMEMAIN.0 \0`. The `\0` is important as it disables some handshaking.
6. It takes several minutes to download the approximately 550 lines of Motorola S-record executable file.
7. When the Kermit prompt appears, type `c` to connect to the EVB again.

8. Type the keys `control-A` simultaneously followed by a carriage return. The ">" prompt should reappear.

9. One can now either type `g c000` or replace jumper J4 and push the restart button on the EVB. Either action will start the famemain program which has a starting address of 0x1800.

To determine whether the automatic startup is enabled on your EVB, type, at the Buffalo prompt, >, `md b600` followed by a carriage return

Buffalo should respond with several lines containing the contents of the memory starting at location 0xB600. The first line should contain `7E 18 00`.

This can only be programmed by running the writejmp.0 program since it is in EEPROM. If automatic startup is not enabled on your EVB, read the file wjread.doc for information on running the program writejmp.0 on your EVB.

The procedure in PROCOMM is similar except the file is uploaded using the page-up key, ASCII transfer (menu number 7), and then typing the filename. When finished, the Buffalo prompt will appear and there is no need to type control-A. On some machines, PROCOMM is terribly slow in uploading and Kermit is much faster.

Remember that FAMEMAIN operates through the upper DB-25 connector at 9600 baud, no parity, 1 stop bit, using a null modem and expects to talk to FAMEPC.EXE.[4]

D.2.5 FAME Software Release 6.0, November 20, 1992.

D.2.5.1 Modifications[6]. FAMEPC no longer controls the helicopter via the PC's keyboard, but rather via the game port using Joysticks provided by the Skylark R/C Helicopter Flight Simulator software package. FAMEPC also can be triggered to collect input/output data

[4]Again note, after the baud rate modification, FAMEMAIN now operates at 19200 baud, no parity, 1 stop bit, without a null modem through the lower DB-25 connector.

[6]This modification has been installed on the AFIT FAME apparatus.

for flight of the helicopter. It is expected to return the keyboard control of the helicopter with the next version of software.

Communications now occurs at 19.2 kBaud over the ACIA port, rather than the SCI port. This allows for measurement of the helicopter position and control based upon this position at a fixed rate of 50 Hz. This upgrade requires a few minor hardware modifications.

1. Communications to the EVB from the PC now takes place only using the bottom port(when looking at the board upside down, with the helicopter up side up), without a null-modem. After downloading FAMEMAIN.0 to the EVB using kermit or some other terminal program, shutting off power to the EVB, moving J4 back to its original position, the serial cable will remain attached to the bottom DB-25 of the EVB.
2. In order to communicate at 19.2 kBaud over the ACIA, a jumper must be made on the board. This jumper must go from pin 9 of U13 to any one of the pins on the left row of pins on J5. Also, the blue jumper must be removed from this row of pins. Note that after this modification is made, all communications (including the downloading of new software) between the PC and the EVB over the ACIA must be performed at 19.2 kBaud.

Full control of the servos is now available with each control string. Incremental control is no longer being used as it was in Version 5.

There is no longer a protocol analyzer available for this format of communication.

D.2.6 Release 5.0, August 25, 1992.[7]

D.2.6.1 Modifications. It appears that all jittering problems have been alleviated. The interrupt for measuring the response from the Gyro seems to have caused jittering in the previous versions when it would be synchronized with one of the pulse widths sent to control a

[7]The following documentation on releases 2.1 through 5.0, while not current is still useful since many of the features are incorporated into version 6.0.

servo. The pulse widths are now generated in such a way that the response from the servo (which triggers the input capture) does not interfere with the output compares.

Jitters caused by the receive interrupt have also been alleviated. The receive interrupt has been disabled, and transmissions from the PC are monitored directly from the foreground. This is possible since the servo control information has been reduced to one byte. Requests to increase or decrease each servo by one out of 250 different settings are sent using a ternary code. With five servos, this system uses the 8 bits to code 0 through 242 ($3^5 = 243$) to control the servos. The Elevator occupies the LSB, followed by the Aileron, Throttle, Collective, and the Rudder is in the MSB. Hence the format shown in Table 1.

Table 1. Servo Control Format

Control	Macro	Code	Control	Macro	Code
Decrease All	ALL_DEC	0	Increase All	ALL_INC	242
No Change	ALL_NO_CHANGE	121			
Dec. Elevator	ELEVATOR_DEC	120	Inc. Elevator	ELEVATOR_INC	122
Dec. Aileron	AILERON_DEC	118	Inc. Aileron	AILERON_INC	124
Dec. Throttle	THROTTLE_DEC	112	Inc. Throttle	THROTTLE_INC	130
Dec. Collective	COLLECTIVE_DEC	94	Inc. Collective	COLLECTIVE_INC	148
Dec. Rudder	RUDDER_DEC	40	Inc. Rudder	RUDDER_INC	202

Further, the numbers above 242 have been allocated to perform the following tasks shown in Table 2.

Table 2. Data Format

Control	Macro	Code
Stop Throttle	REQ_DEFAULT_THROTTLE	243
Set all Servo Values to Default	REQ_DEFAULT_SER_VALS	244
Dummy Value – EVB Internal Use	DUMMY	249
Send Potentiometer Values	POT_REQ	254
Send Servo Values/Settings	SER_REQ	255

The EVB now acknowledges all servo control bytes (0 - 242) with the potentiometer values. The potentiometer values consist of 9 bytes: 6 for the 6 potentiometers, 2 for the yaw gyro and 1 for the checksum. The Start, Command, and Stop characters are no longer included. With this change comes a change in the sequence of error measurement. Instead of the sequence listed under the revision notes for version 4.0, below, the sequence is shown in Table 3.

Table 3. Control Sequence

	PC:	EVB:	PC:	PC:	PC:	PC:	
	Send	Ack w/	copy	Calc	Calc	Calc	
...	servo	Pot	old	new	error	servo	...
	values	values	error	error	dot	values	
		vector	vector	vector			

The new rate of this sequence is 66 Hz.

D.2.6.2 FAMEPC.

Dynamic Control. FAMEPC now Dynamically Controls with a Proportional, Integral, Double Integral controller. While in the Dynamic Control mode and after choosing a particular servo for which to vary the transfer coefficients, the keyboard is laid out as shown in Table 4.

Table 4. Keyboard Layout

Info:		(F1)	(F2)	(F3)	(F4)	(F5)	(F6)	(F7)
Double Int	Inc:	(1)X	(2)Y	(3)Z	(4)Pitch	(5)Roll	(6)Yaw	(7)Yaw Gyro
	Dec:	(Q)X	(W)Y	(E)Z	(R)Pitch	(T)Roll	(Y)Yaw	(U)Yaw Gyro
Integral	Inc:	(A)X	(S)Y	(D)Z	(F)Pitch	(G)Roll	(H)Yaw	(J)Yaw Gyro
	Dec:	(Z)X	(X)Y	(C)Z	(V)Pitch	(B)Roll	(N)Yaw	(M)Yaw Gyro
Proportional	Inc:	(a)X	(s)Y	(d)Z	(f)Pitch	(g)Roll	(h)Yaw	(j)Yaw Gyro
	Dec:	(z)X	(x)Y	(c)Z	(v)Pitch	(b)Roll	(n)Yaw	(m)Yaw Gyro
(P)otentiometer Data ([)Position Data (-)Error Vector (8)Display PC Servo Vals (9)Display EVB Servo Vals (0)Equalize Gyro Offset: (F8)Display (F9)Decrement (F10)Increment (I) quit varying values for this servo								

The error vector can now be displayed real time by typing the (-)minus dash key. Also, there are some cases where the servo values maintained by the PC become different from those of the

EVB. (8) Displays the PC's servo values, (9) request the servo values from the EVB and displays them, and (0) Requests values from the EVB and sets the PC's values equal to these requested values.

When setting the Transfer Coefficients for the Yaw Control, it is often helpful to be able to modify the Gyro Offset value, since it is this value that serves as the DC offset of the yaw from the desired value. It has been found that the dynamic control feature is able to keep a specified yaw setting; however, this yaw setting will always have a non-zero error as revealed using the (-) key. This error can be set to zero by adjusting the Gyro Offset value. Thus (F8) displays the current setting, (F9) decreases and (F10) increases the Gyro Offset value.

At George Mason University, we have achieved good control of the Pitch, Roll & Yaw axis and are currently searching for transfer coefficients to control the elevation. The GENERIC.COE file that is a part of this package reflects these attempts. After the elevation is stabilized, I expect we will work on the X & Y axes.

The control routines are in the function DynamicControl(), and are taken from *Discrete-Time Control Systems* by Katsuhiko Ogata, 1987, pp. 200-204. The nature of the communications link between the PC & EVB requires that control be in the velocity form, that is, instead of directly calculating values for the servo settings, values are calculated to increment or decrement the servo settings.

Control of the helicopter was first attempted with a discrete PID control, but we were unable to achieve reliable control of the pitch and roll axes. Next, we attempted control using a P-I-II controller, and were able to achieve stability. The P-I-II control was formed by applying the positional form of control to the helicopter

In PID control, the positional form(direct control) is

$$
\begin{aligned}
u(t) &= K_i e(t)dt + K_p e(t) + K_d \frac{de(t)}{dt} \\
e(t) &= \textit{desired position} - \textit{actual position vector}
\end{aligned}
$$

The discrete version of this is

$$
\begin{aligned}
u(t) = {} & K_i \left[\frac{(e(0) + e(T))}{2} + \frac{(e(T) + e(2T))}{2} + \cdots + \frac{(e(k-1) + e(k))}{2} \right] + \\
& K_p e(k) + K_d e(k) - e(k-1)
\end{aligned}
$$

For the integral term, the program keeps error vectors for four terms into the past. Also, the divide by two for each pair of error vectors is not included since this would merely scale the Ki coefficient. Because we are applying the positional form to a problem that requires a velocity form, our control function looks like:

$$
\begin{aligned}
u(t) = {} & K_{ii}[e(k-4) + e(k-3) + e(k-3) + e(k-2) + e(k-2) + e(k-1) + e(k-1) + e(k)] + \\
& K_i e(k) + Kpe(k) - e(k-1)
\end{aligned}
$$

D.2.6.3 FAMEMAIN. All calibration of potentiometer values now takes place on the PC. The commands to request Position values directly from the EVB have been removed, since this is a time consuming process, and it is difficult to create the calibration coefficients on the EVB. This software release does not contain the code for FAMECAL, which is now obsolete. Also, the code in the file FAMEINI2.c has been copied into FAMEINIT.c since it is no longer necessary to distinguish between initialization functions required for both FAMEMAIN & FAMECAL, and those that are only used in FAMEMAIN.

D.2.6.4 Protocol Analyzer. In order to view the commands being sent to the EVB real time, a 'protocol analyzer,' PROTOAN.EXE was developed. The RX & Ground lines of a second PC's serial port are connected to the TX & Ground lines (respectively) of the PC running FAMEPC. This program decodes the byte sent to the EVB and displays 5 columns of -1, 0, or 1 to indicate the servo controls sent. They are in the same order as listed above: Elevator, Aileron, Throttle, Collective, Rudder.

D.2.7 Release 4.0, July 20, 1992.

D.2.7.1 Modifications. This version release marks a significant change in the FAME software, both structurally and in the features offered. There have been complaints and we have experienced 'jittering' in the servos which seems to be caused by the HC 11. We hypothesized that this was caused by interrupts for servo control being unable to be triggered because interrupts for communication were active. Hence, we have moved the non-time critical communications processing code from the background (from the ISRs) to the foreground (to the main() function) where it can get interrupted.

This has solved some of the jittering problem; however, there are still instances of gyros jittering. For instance, our Aileron jitters even when no communications is taking place. We have some ideas as to the resolution of this problem and will announce another software release when we resolve it.

Another modification made to reduce jittering and reduce the amount of time spent controlling the Helicopter was the reduction in the size of the Servo Values String. Previously, this string used two bytes for each servo value transmitted to the HC 11; this has been reduced to one byte per servo. Hence, the Servo Value String uses 9 instead of 14 bytes.

D.2.7.2 FAMEPC. Code has been developed for implementing a PD controller for the Helicopter. Upon pressing the tilde (~) key while in the Keyboard Servo Values mode, FAMEPC begins by establishing a vector describing the desired position of the helicopter. The program measures the present position of the helicopter by sending out a potentiometer value request and converting it in into a Cartesian position based on its Calibration Coefficients. The measurements of X, Y, Z, & Yaw from this request become a part of the vector of desired values. Entries in this vector for Pitch, Roll & the Yaw Gyro are set to 0, since it is desired to have the Helicopter flying balanced and with a non-existent Yaw rotation rate.

Having established a desired position vector for the helicopter, FAMEPC continues by repetitively sending potentiometer value requests. These potentiometer requests are also converted into a Cartesian position & gyro vector by FAMEPC, and are then used to establish an error and rate of error (error dot) vector. The error values are found by measuring the difference between the desired position vector & the actual position vector. The error dot vector is calculated after every second potentiometer request by finding the difference between the previous two error vectors.

The potentiometer request that occurs after the error dot vector is calculated, that is, the first of the set of two potentiometer requests, also contains servo control values. These servo control values are calculated by updating the previous servo control values with changes calculated from a matrix that is multiplied by the error & error dot vectors. Valid values for this matrix are still being measured by trial and error at George Mason University.

This transfer matrix can be edited in a variety of manners. From the main menu, an option is available to directly edit it. Also from this menu, one can edit real time change values for each of the coefficients. These change values are amount the coefficient will change when a particular key is pressed during real time control operations. Only the coefficients which constitute inputs to one servo can be varied at any given instant.

While real time control is taking place, the set of error & error dot coefficients to be varied for a particular servo are selected by typing the first letter of that servo: (E)levator, (A)ileron, (T)hrottle, (C)ollective, or (R)udder. From this point, incrementing or decrementing a transfer value takes place by typing the key from the Table 5.

Table 5. Control Keys

Info:	(F1)	(F2)	(F3)	(F4)	(F5)	(F6)	(F7)
Inc:	(1)X	(2)Y	(3)Z	(4)Pitch	(5)Roll	(6)Yaw	(7)Yaw Gyro
Dec:	(Q)X	(W)Y	(E)Z	(R)Pitch	(T)Roll	(Y)Yaw	(U)Yaw Gyro
Inc:	(A)Xdot	(S)Ydot	(D)Zdot	(F)PitchDot	(G)RollDot	(H)YawDot	(J)Yaw GyroDot
Dec:	(Z)Xdot	(X)Ydot	(C)Zdot	(V)PitchDot	(B)RollDot	(N)YawDot	(M)Yaw GyroDot
(I) quit varying values for this servo					(O)display servo value settings		

The Esc key works as usual to stop the throttle & break the real time control mode. Pressing a function key for information provides data about the value of the matrix coefficients & the change rates for the particular degree of freedom.

The transfer matrix of control coefficients and the rates at which they should be varied can be saved to and loaded from a DOS file. Every time FAMEPC is executed from DOS, the file generic.coe is loaded into memory for the transfer matrix & varying values, and upon quitting FAMEPC, the current values are saved to this file. Analogous operations occur with the file generic.cal for the calibration coefficients.

D.2.8 Release 3.0, June 23, 1992.

D.2.8.1 Modifications.

FAMEPC.

Repetitively Vary Servo Control Values. The keyboard layout for varying servo control values has been changed to a format similar to that of a radio control unit. The Escape key now sets the throttle to its lowest level, acting as an emergency stop key. When the throttle is at its lowest level, striking the equals sign or plus sign key will set the servo control values to their defaults. Further, visual indications of the servo values have been removed and a tone is sounded after each keystroke. When the limit has been reached, an extremely low or high tone is sounded instead.

In the previous version, servo control values were repetitively sent to the EVB as a part of the loop that checked for keystrokes, that is, they were sent irregardless of whether a change was requested at the PC's keyboard. Now, the servo control string is sent to the EVB only after a keystroke to change the servo values occurs.

Measure Step Response. The measure step response option sends the specified step impulse to the EVB and saves data to the specified file every time the I key is struck. The Q key should be struck to exit the measure step response option. The time measurements are recorded in milliseconds, and are based upon the time that the step occurs, such that the five preceding samples have negative time values.

Calibration. Calibration values now can be downloaded from the EVB, measured with the PC, saved to the PC's disk, and be loaded from disk for providing Cartesian & Attitude location using the PC's computational power, rather than the EVB. This results in a significant savings in time.

FAMEMAIN. Code now exists to transmit Calibration Coeffients to PC upon receipt of request as noted below in Revised Message Format Chart (Table 6).

FAMECAL. Before saving calibration coeffients to SRAM, their values are displayed so that the user can decide if they are appropriate.

FAMEPC, FAMEMAIN, & FAMECAL. A number of variables' names were changed to make the programs more consistent.

D.2.9 Release 2.1, May 27, 1992 Bugs Fixed.

Table 6. Revised Message Format Chart

Key: S = Start Char, s = Stop Char, _ = Servo Control Value, % = Checksum		
Origin	Message	Message String
PC	Control Servos	SC__________%s (chars 2-11 servo con. vals.)
PC	Req. Potentiometer Vals.	SQs
PC	Request Position Values	SRs
PC	Control Servos & Req. Potentiometer Vals.	SS__________%s
PC	Control Servos & Request Position Values	& ST__________%s
PC	Request Calibration Coef	SYs
EVB	Ack of Servo Control	SAs
EVB	Potentiometer Values	SO,AZpot(2),ROLLpot(3),ELpot(4),YAWpot(5), Hpot(6),PITCHpot(7),YawDot(8-9), Time Stamp(10-12),%s
EVB	Position Values	SP,X(2-3),Y(4-5),Z(6-7),Roll(8-9), Pitch(10-11),Yaw(12-13),YawDot(14-15),%s
EVB	Calibration Coefficients	SV,AZslope(2-5),ROLLslope(6-9), ELslope(10-13), YAWslope(14-17), Hslope(18-21),PITCHslope(22-25), AZdcOffset(26),ROLLdcOffset(27), ELdcOffset(28), YAWdcOffset(29), HdcOffset(30),PITCHdcOffset(31),%s

D.2.9.1 FAMEPC. FAMEPC can now be changed between COM 1 & COM 2. Previously the command to select COM 2 wouldn't work.

The code for awaiting acknowledgement from the EVB has been enhanced. The default delay is 0 ms between transmissions, and it no longer has any relation to the amount of time that FAMEPC will wait before timing out due to a NAK (no acknowledgementΦ) from the EVB. FAMEPC is internally set to time out if it doesn't receive a response from the EVB after 3 seconds.

Version 2.0 of FAMEPC concatenated position integers from the EVB into unsigned values. This generated erroneous values for the position measurements sent from the EVB. Version 2.1 corrected this bug.

D.2.9.2 Modifications.

OVERALL. The FAME package now supports the measurement of raw (uncalibrated) potentiometer measurements, and FAMEMAIN will also send position or potentiometer data to the PC as a part of a Servo control request. The new chart of PC/EVB Message formats is in Table 7.

Table 7. New Chart of PC/EVB Message Formats

Key: S = Start Char, s = Stop Char, _ = Servo Control Value, % = Checksum		
Origin	Message	Message String
PC	Control Servos	SC__________%s (chars 2-11 servo con. vals.)
PC	Req. Potentiometer Vals.	SQs
PC	Request Position Values	SRs
PC	Control Servos & Req. Potentiometer Vals.	SS__________%s
PC	Control Servos & Request Position Values	& ST__________%s
EVB	Ack of Servo Control	SAs
EVB	Potentiometer Values	SO,AZpot(2),ROLLpot(3),ELpot(4),YAWpot(5), Hpot(6),PITCHpot(7),YawDot(8-9), Time Stamp(10-12),%s
EVB	Position Values	SP,X(2-3),Y(4-5),Z(6-7),Roll(8-9), Pitch(10-11),Yaw(12-13),YawDot(14-15),%s

Upon requesting the Potentiometer data using FAMEPC, the user receives an 8 bit response from the 6 potentiometers, a 16 bit measurement of yaw dot, and a 24 bit value of the internal clock on the EVB sampled at the end of the function that triggers the A/D converter. The actual clock consists of 16 bits; however, an ISR has been added that increments an 8 bit variable at each overflow of the timer.

FAMEPC has been upgraded to measure a step response from the helicopter. This routine prompts the user for the file name in which to store the data, for which Servo to provide the step (Throttle, Aileron, Elevator, Rudder or Collective), and for the change in the value of the specified servo.

Next, the routine places the user into the "Servo Control with Keyboard" mode so that the user can adjust the helicopter to the desired position. Upon pressing the (I) key, FAMEPC requests 5 potentiometer measurements, changes the value of the specified servo, and requests 16 more potentiometer measurements. This data is stored into the PC's memory in real time, and is transferred to the file after the impulse is measured. An asterix is placed next to the measurement that occurs immediately after the step response is executed. The file also contains the information specified by the user preceding the step pulse, as well as the setting of the Extra delay between requests value.

We at George Mason intend to measure the first step responses from the aileron and elevator by keeping the helicopter tied down with a vise, thus paying particular attention to the pitch and roll measurements.

TIME SPANS. One cycle of a position request & response takes 150 ms. This consists of:

PC TX to EVB	A/D conv,cal	EVB TX to PC	PC display data
3 ms	95 ms	18 ms	37 ms

One cycle of the new potentiometer request & response uses only 60 ms.

PC TX to EVB	A/D conv.	EVB TX to PC	PC display data
3 ms	2 ms	15.5 ms	39.5 ms

Darrell Duane

dduane@fame.gmu.edu

George Mason University Electrical & Computer Engineering

November 20, 1992

D.3 Report on the Fast Adaptive Maneuvering Experiment (FAME)

A copy of the Report on the Fast Adaptive Maneuvering Experiment (FAME) can be found accompanying the AFIT FAME apparatus. The report is also available in PostScript format via anonymous FTP to "fame.gmu.edu". It is strongly advised to read the entire report along with all supporting documentation prior to any attempts at manual flight control.

D.3.1 Changes to FAME Report. The following are a list of changes and addenda to the report as a result of the experimentation conducted in the course of the research for this thesis.

- page 9, section 2.2, line 3. "There are two switches on the plate which supports the helicopter." These switches are now attached to the landing strut of the helicopter. The switches were moved to permit easy removal of the helicopter for maintenance and free-flight.
- page 10, section 2.2.1. Additional Comment: The rudder control servo, located directly below the speed controller on the helicopter, has been repositioned so that the servo shaft is located on the opposite side of the helicopter. This was necessary due to a rudder control reversal.
- page 13, section 2.3.2. Additional comment: A 9.6 VDC trickle charger has been purchased in the course of this research. It will fully recharge the Nicad battery pack in 12-16 hours. Also, three 12 VDC Gel-Cell batteries are also available for use. It is important to keep the Gel-Cells fully charged, especially when storing the batteries for long periods of time.
- page 19, section 3.9. The red collar has been modified so that it will completely support the helicopter when not flying and will drop away as the it gains altitude. This change helps reduce adverse yaw as the motor is run up to flight speed.
- page 20, section 3.9. Fourth paragraph: A metal stop has been added to the yaw potentiometer mount to limit the yaw range of motion. As delivered, the yaw range of motion exceeded the yaw potentiometer range. The yaw range is approximately ±90°.
- page 20, section 3.9. Last paragraph: There is not screw/nut pair on the collar.

- page 21, section 3.10. Connectors have been added to permit quick and easy removal of the helicopter from the flight stand. These connectors are also standard R/C type to permit connection to a standard R/C receiver for free-flight control. Also, the helicopter has been secured to the stand using plastic wire wraps, again to permit easy removal.
- page 23, section 5. The primary point of contact for FAME at George Mason University is presently Darrel Duane, E-mail dduane@mason1.gmu.edu.
- appendices 6.1 through 6.4. The supporting software for FAME has changed several times since the report was issued. If desired, the code for FAMEPC versions 2.1 through 6.0 is available through FTP with "fame.gmu.edu." Appendix A contains the modified code listing used in this research.
- appendix 6.5. Additional drawings are provided documenting the DOF collar modification.

D.4 AFIT FAME User's Guide

D.4.1 Introduction. The following provides a brief overview of the setup and operation of the AFIT Fast Adaptive Maneuvering Experiment (FAME) apparatus. Prior to any attempt at operating the FAME apparatus, recommend a thorough review of the following documents:

1. Report on Fast Adaptive Maneuvering Experiment (FAME) ((5))
2. Kalt Electric Helicopter Baron Whisper Instruction Manual (7)
3. SKYLARK R/C Helicopter Flight Simulator Users Manual (3)
4. Ray's Complete Helicopter Manual R/C (6) (optional)

D.4.2 Hardware. The following is a list of hardware items for the AFIT FAME apparatus:

- 386-based (minimum) PC with at least one comm port
- Skylark controller and I/O card

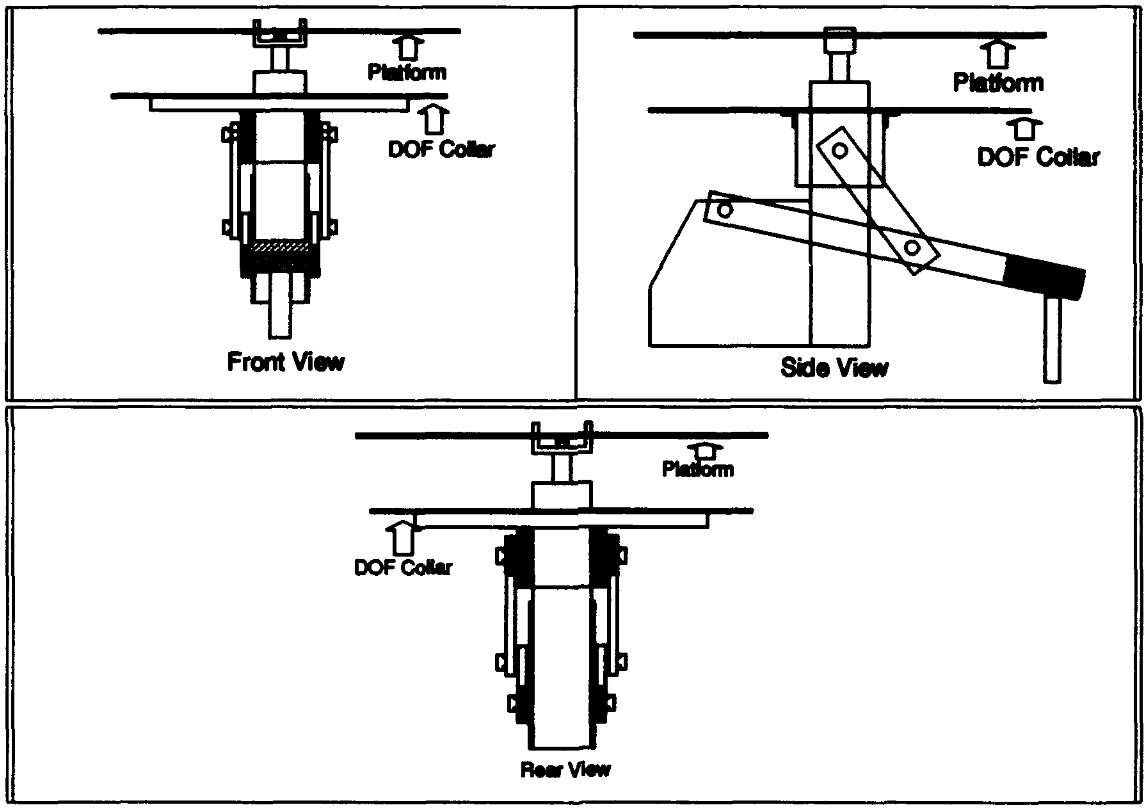


Figure 24. DOF Collar Modification (Not to Scale)

- FAME apparatus
  - KALT Whisper Electric Helicopter
  - Flitemaster Jr. Training Stand (instrumented)
  - Nicad Battery, 1350 mAh
  - Motorola MC68HC11 microcontroller unit (MCU)
  - MCU Power Supply
- One (1) 9.6V Nicad Recharger, Radio Shack
- Three (3) 12V Power Sonic Gel-Cell Batteries
- Two (2) 25 lb concrete counterweights

D.4.3 Software Installation. The following is a list of the MS-DOS software necessary to maintain and operate the FAME:

- Operation
  1. MS-DOS 2.0 or higher
  2. FAMEPC 6.0 (modified 12 Oct 93)
  3. GENERIC.CAL
- Maintenance
  1. FAMEMAIN.0
  2. WRITEJMP.0
  3. PROCOMM or some other communications program
  4. Borland C or another MS-DOS based C-compiler

FAME will run from a floppy, but if possible, install FAME on the PC harddrive.

The MS-DOS file "generic.cal" must also be included in the same directory. This file provides values for deriving state values from potentiometer readings. Without this file, the state values will alway read zero.

D.4.4 Operational Instructions. To start the FAME program, at MS-DOS prompt type **FAMEPC**. The FAME menu should appear (Figure 25).

D.4.4.1 Menu Selection Summary.

- (J) - allows joystick control of the helicopter, but does not record the state or servo values.
- (G) - activates control of the helicopter using the joystick and stores the servo and state values in the file "file1.trn". Recording time is limited to approximately thirty seconds. If allowed to record for a longer period, the program empties the file and no longer records data. Also, any subsequent recordings will overwrite the data currently in "file1.trn". To save the data, after each recording rename "file1.trn." The format of "file1.trn" as read left to right: time(sec), elevator, aileron, throttle, collective, rudder, pitch, roll, altitude, y-coordinate, yaw, x-coordinate, and yaw-dot.
- (C) - calibrates the joystick and stores values in "joy.cal".
- (S) - Displays realtime state values (pitch, roll, alt, y-coord, yaw, x-coord, yaw-dot).
- (X) - Display realtime raw potentiometer values (H pot, Az pot, El pot, pitch, roll, yaw, gyro).
- (Q) - Exit program

D.4.5 Flight Operation.

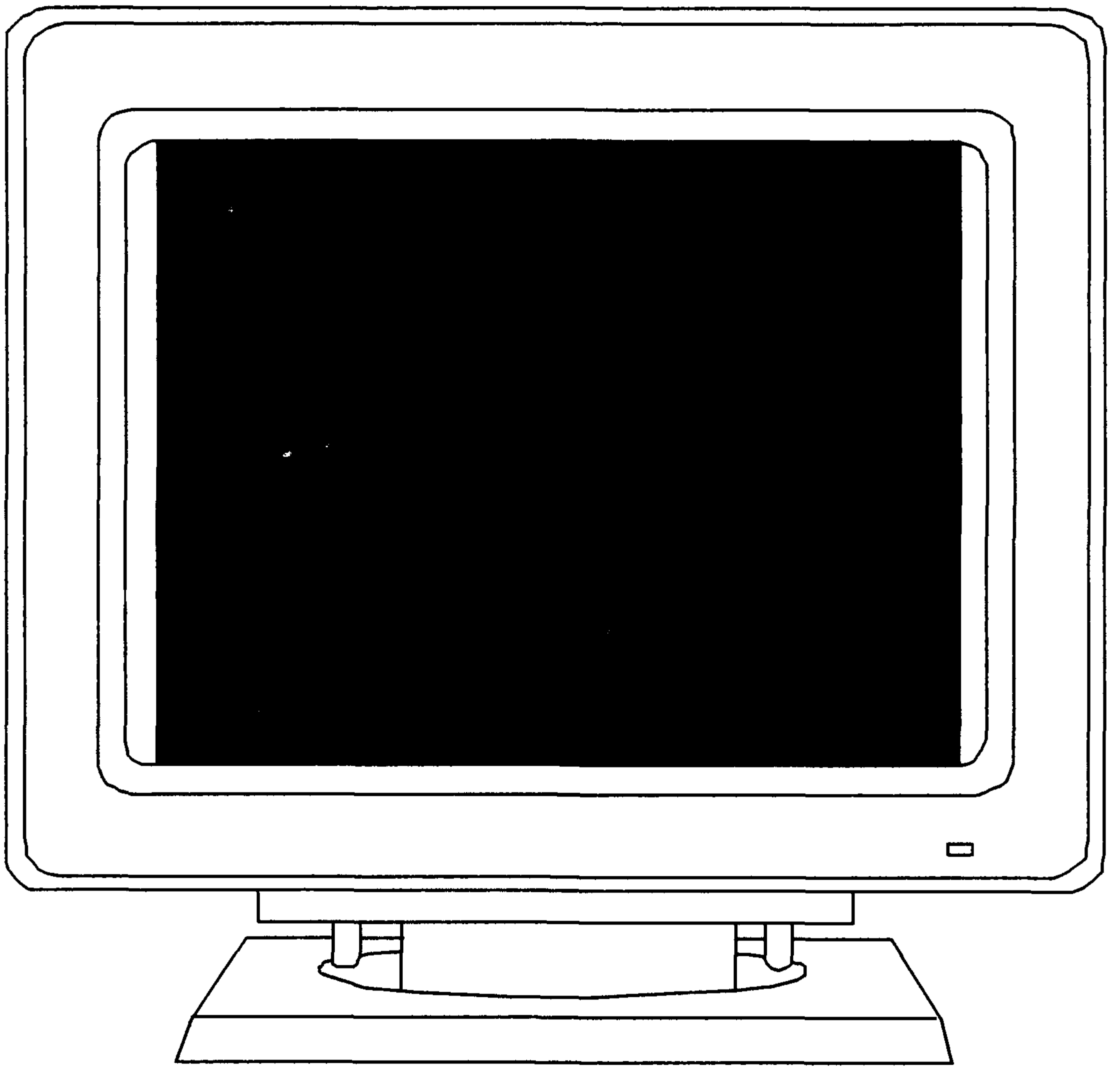


Figure 25. FAMEPC Main Menu

<u>WARNING</u>. Although small and battery powered, the Kalt Helicopter is not a toy. Strongly advise reading all documentation on the Kalt Helicopter, FAME apparatus, with special attention paid to the warning and safety recommendations. Prior to attempting to operate the FAME helicopter, also strongly recommend spending time with the SKYLARK simulator program. It is also advisable to spend a few hours of instruction with an experienced R/C helicopter pilot.

The following is a step-by-step guide to the operation of the AFIT FAME apparatus. It is only a guide and is not intended to stand alone. The user should still read all accompanying documentation on FAME and the AFIT FAME apparatus.

1. Ensure flight area is clear of all obstacles within the flight radius of the FAME apparatus.

2. Place the FAME apparatus on a smooth surface (floor). Place counterweights on the plywood platform opposite the helicopter. Also place the MCU power supply and the helicopter power supply (batteries) on the platform. Connect the RS-232 to the PC. Connect all power. By hand, move the training stand through all range of motion while checking for binding or wires caught in the training stand joints.

3. To prevent accidental throttle application, **<u>DO NOT</u>** activate the motor (press the red button) while the FAME program is in joystick control (J) or (G) at this point. When starting joystick control, ensure the throttle is set at lowest level (full down).

4. Turn on all power to the PC, MCU and helicopter. **<u>DO NOT</u>** press the red button near the helicopter power switch at this time. A high pitched whine should be heard from the helicopter gyro and the LED indicator on the motor controller should be green. If not, then power is not being provided to the helicopter. Check power leads and the 30 Amp fuse on the motor controller.

5. Start FAMEPC by typing "`famepc`" at the MS-DOS prompt. Check communications to the MCU by selecting (S) or (X). The appropriate state or potentiometer values should be seen. If not, power down and recheck all connections.

6. If joystick has not been previously calibrated, select (C) and follow instructions to calibrate joystick. The calibration data is stored on disk in "joystick.cal". Calibration will not be necessary later.

7. Select (J) or (G). Check flight control by moving the joystick controls and observing servo movements. Check throttle by observing LED readout on the motor controller. LED should change from green to yellow then red as the throttle is increased. Hit any key to terminate joystick control.

8. Ensure the area around the helicopter rotor is clear. Check that the throttle is set at lowest position (full down). Select (J) or (G) to activate joystick control. Check the LED readout on the speed controller. If green, it is safe to enable the motor by pressing the red button on the helicopter. Move to a safe distance and slowly apply throttle while observing the main and tail rotors. Throttle down immediately if any unusual vibration occurs or the rotors strike an object. If helicopter begins to perform erratically or becomes uncontrollable, immediately power down the helicopter by disconnecting the battery or DC power supply. Alternately, turn of the MCU power supply. This will cause all servos and the motor controller to return to the neutral setting.

9. To end flight control, slowly move the throttle to the lowest position. After all movement has stopped strike any key to terminate joystick control. Place main power switch on helicopter in the "off" position.

10. If subsequent flights are to be recorded, rename "file1.trn" or the data recorded will be overwritten by the newer data.

D.4.6 Troubleshooting Guide. The following table summarizes some of the more common problems encountered when operating FAMEPC.

Table 8. Troubleshooting Guide

Problem	Possible Cause	Solution
Receiving NAKs after selecting (S) or (X)	No data path	Check RS-232 connection
	No power to MCU	Check power to MCU
	Wrong port	Check COM1
	MCU memory erased	Reload FAMEMAIN.0 and/or WRITEJMP.0
	Jumper J4 on MCU in wrong position	Check J4 in 2-3 position
Data file fails to record	Disk is full	Remove files as necessary
	Flight time was longer than 30 seconds	Record data again over shorter time

Bibliography

1. Antsaklis, Panos J. "Neural Networks in Control Systems," *IEEE Control Systems Magazine, 12* (April 1992).
2. Calvin, James B., Jr. *ETANN Hardware Implementation for Radar Emitter Identification.* MS thesis, Air Force Institute of Technology (AETC), 1992.
3. digital Wonder Works, Inc., P.O. Box 3118, Ann Arbor, MI 48106. *Skylark R/C Helicopter Flight Simulator User's Manual, Version 1.03.*
4. Hertz, John and others. *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley Publishing Company, 1991.
5. Hintz, Kenneth J. *Report on the Fast Adaptive Maneuvering Experiment (FAME).* Technical Report AFOSR-91-0372, George Mason University, March 1992.
6. Hostetler, Ray. *Ray's Complete Helicopter Manual* (3rd Edition). Sierra Madre, CA: R/C Modeler Corporation, 1991.
7. Kalt Sangyo Co., Ltd., 1447-1 Higashi-Tanaka, Gotenba, Shizuoka 412, Japan. *Kalt Electric Helicopter Baron Whisper Instruction Manual.*
8. Lehr, Michael. *Adaptive Multisource Decision-Making.* Technical Report BRDE-ISL/TR-1/1, Stanford University, April 1992.
9. Lindsey, Randall L. *Function Prediction Using Recurrent Neural Networks.* MS thesis, Air Force Institute of Technology (AETC), 1991.
10. Magnus, Jan R. and Heinz Neudecker. *Matrix differential Calculus with Applications in Statistics and Econometrics.* New York: John Wiley and Sons, 1988.
11. Narendra, Kumpati S. and Snehasis Mukhopadhyay. "Intelligent Control Using Neural Networks," *IEEE Control Systems Magazine, 12* (April 1992).
12. Nguyen, Derrick H., "Re: NN for Self-Learning Adaptive Control." Electronic mail received October 28, 1993.
13. Nguyen, Derrick H. and Bernard Widrow. "Neural Networks for Self-Learning Control Systems," *IEEE Control Systems Magazine, 10* (April 1990).
14. Prouty, Raymond W. *Helicopter Performance, Stability, and Control.* Malabar, FL: Robert E. Krieger Publishing Co., Inc., 1990.
15. Robinson, Anthony John. *Dynamic Error Propagation Networks.* PhD dissertation, Cambridge University, 1989.
16. Rogers, Steven K. and others. *An Introduction to Biological and Artificial Neural Networks.* Air Force Institute of Technology (AETC), 1990.
17. Scharf, Louis L. *Statistical Signal Processing, Estimation and Time Series Analysis.* Reading, MA: Addison-Wesley Publishing Company, 1991.
18. Schauf, Charles L. and others. *Human Physiology, Foundations and Frontiers.* St. Louis, MO: Times Mirror/Mosby College Publishing, 1990.
19. Schley, Charles and others. "Neural Networks Structured for Control Applications to Aircraft Landing." *Advances in Neural Information Processing Systems 3* edited by Richard P. Lippman and others, 415–421, San Mateo, CA: Morgan Kaufmann Publishers, 1991.
20. Seddon, J. *Basic Helicopter Aerodynamics.* Washington, DC: American Institute of Aeronautics and Astronautics, Inc., 1990.

21. Tarr, Gregory L. *Multi-Layer Feedforward Neural Networks for Image Segmentation.* PhD dissertation, Air Force Institute of Technology (AETC), 1991.

22. Widrow, Bernard and Jr. M.E. Hoff. "Adaptive Switching Circuits." *1960 IRE WESCON Convention Record, Part 4*. 1960.

23. Wu, Q. H. and others. "A Neural Network Regulator for Turbogenerators," *IEEE Transactions on Neural Networks*, *3*(1) (January 1992).

Vita

Captain Ronald E. Setzer was born June 14, 1962 in Tokyo, Japan. After graduating from Lincoln County High School in Panaca, Nevada, he was accepted to and attended the United States Air Force Academy in Colorado Springs, Colorado. Graduating in May 1984 with a Bachelor of Science in Electrical Engineering, he was assigned to duty at the 2nd Combat Communications Group in Patrick AFB, Florida. In 1986 he was transferred to the 4th Combat Communications Squadron in Yokota AB, Japan. While stationed in Japan, he returned to Florida to complete his Masters Degree in Business Administration, awarded in December 1987. In November 1988, he was assigned to Operating Location AH, Communications System Program Office, McClellan AFB, CA. In February 1991, he was transferred to Detachment 16, Electronic Systems Division, better known as the Peace Shield Site Activation Task Force, Riyadh, Saudi Arabia. In April 1992, he left Saudi Arabia for the Air Force Institute of Technology at Wright-Patterson AFB, Ohio. Captain Setzer is unmarried and has no children.

Permanent address: P.O. Box 322
Panaca, NV 89042

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Neural Networks for Dynamic Flight Control	5. FUNDING NUMBERS
6. AUTHOR(S) Ronald E. Setzer, Capt, USAF	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/93D-36
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Capt Steven Suddarth AFOSR/NM 110 Duncan Ave, Suite B115 Bolling AFB DC 20332-6448	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis examines the application of artificial neural networks (NNs) to the problem of dynamic flight control. The specific application is the control of a flying model helicopter. The control interface is provided through a hardware and software test bed called the Fast Adaptive Maneuvering Experiment (FAME). The NN design approach uses two NNs: one trained as an emulator of the plant and the other trained to control the emulator. The emulator neural network is designed to reproduce the flight dynamics of the experimental plant. The controller is then designed to produce the appropriate control inputs to drive the emulator to a desired final state.

Previous research in the area of NNs for controls has almost exclusively been applied to simulations. To develop a controller for a real plant, a neural network must be created which will accurately recreate the dynamics of the plant. This thesis demonstrates the ability of a neural network to emulate a real, dynamic, nonlinear plant.

14. SUBJECT TERMS Neural Networks, Helicopter Control, Fast Adaptive Manuevering Experiment (FAME)			15. NUMBER OF PAGES 140
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL